SIMULATION OF A TANK BATTLE ON COMPUTERS

A Thesis Submitted
in Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

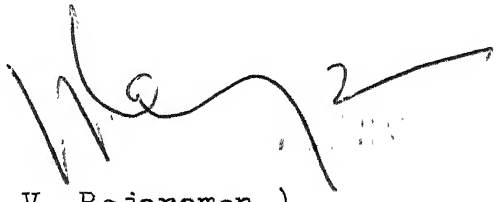
By

Capt. VINAY SAGAR

to the
DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR
SEPTEMBER, 1981

CERTIFICATE

This is to certify that the thesis entitled 'SIMULATION OF A TANK BATTLE EXERCISE ON COMPUTERS' by Capt. Vinay Sagar is a record of work carried out under my guidance and has not been submitted elsewhere for a degree.



Sept. 1981

(V. Rajaraman)
Professor
Electrical Engineering Department
Indian Institute of Technology
KANPUR

ACKNOWLEDGEMENT

I would like to thank Dr. V. Rajaraman, my thesis supervisor for all the help and guidance he has given me. It has given me great pleasure to work under him and get his valuable advice.

I would also like to thank Mr. J.S. Rawat for typing out this thesis neatly.

Capt. Vinay Sagar

ABSTRACT

An interactive simulator for tactical exercise with battle tanks has been described here. The effort has been to make this simulator a logical extension of a sand model exercise, so that a more realistic exercise which is closer to real exercises in field may be achieved.

The topographical details of a map area required to be used, is stored in the memory. The user is permitted to give commands to MOVE and FIRE. The move and fire is simulated taking into consideration the topographical details, characteristics of armour and the firing capability of tanks. The REPORT facility in the simulator helps the user in gaining information regarding tank status and other details used for taking decisions.

It is considered that simulator when fully developed will form a useful aid in exercising a squadron as a team, in which the squadron commander along with his troop commanders can participate.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

In Army, tactical exercises form an important part in troop training. These exercises involve the process of making decisions under visualised situation to resemble true combat situation. Troop commanders are exercised under constraints of carefully planned rules, as they would under combat conditions.

Field Exercises are very costly and have other effects like cost in fuel consumption, wear and tear of equipment and wastage. Wear and tear of tanks is an important factor which reduces the tank life considerably. As such frequent field exercises involving tanks are not possible. Moreover the area available for such exercises is restricted. Built up areas cannot be used for exercise and firing because of the damage it would cause. Also exercise cannot be done in critical areas like borders etc. Therefore at present the use of 'Sand Model' to exercise the personnel in tactics is the most commonly used method. Any form of exercise being at best, a poor substitute for experience, the emphasis is therefore more on realism and greater detail. The objective of this thesis is to develop

a computer based interactive simulation in an effort to replace the sand model exercise with increased reality and more details.

1.1 The Importance of a Simulator:

The simulator comes in between the sand model and field exercise and can have certain advantages over it. Some of the essential features of the simulator are described in the succeeding paragraphs.

The preparation of a sand model is time consuming and even then all the details cannot be represented on it. In constrast in the computer simulation the representation of the topography exists in the memory of the computer which can be more exhaustive than a sand model.

The hypothetical vehicle i.e. tank in this case is 'moved' and can 'fire' taking into account the factors that govern movement such as terrain, gradient, mobility and for fire accuracy of gun,rate of fire and penetration etc.

Visibility and sighting of the tanks is decided by the computer depending on the topographical information available with the computer. Factors that decide visibility like cover,concealment and line of sight is accounted for.

Communication, queries and replies between the simulated tanks by the commanders can be done through the computer.

Freedom of action is available to the participating personnel and pre-determined action is curtailed so that the exercise is allowed to take its course of action.

1.2 Review of Similar work done earlier:

In recent past a lot of thought is going in this direction of simulation on computer mainly because of the economics of time, cost and effort saving that it offers.

Known similar work has been done on 'war gaming' [1], in which a game is described where in two opposing forces are made to 'move' from one point to another according to a battle plan. During this course of movement analysis is done so that if enemy is sighted then it fires a fixed number of rounds/shells on it. In the end the success or failure is evaluated in terms of quantum of forces that have reached their destination.

Another work that has been done has been on 'Computers and Tactical war gaming' [2]. This is a theoretical description on how simulation can be done for a tank battle at squadron level. This work describes how computer can be

used to exercise the commanders i.e. squadron and troop commanders in taking decision in near real life situation. This is more of an interactive process as would be seen in an exercise. The idea for this thesis is based on this work and simulation programmes have been developed in PASCAL for its implementation on Dec 1090.

1.3 Goal of this Thesis:

The goal of this thesis is 'To write the necessary programmes to realise the command and communication reportoire in a higher level computer language for simulation of tank battle'.

To achieve the above goal the following essential features are considered necessary:

a) A detailed description of battle area topography to be stored in the memory. Roads, rivers, obstacles, minefields, untankable country etc. are depicted so that the exercise can be simulated on any desired type of terrain or area for which map is available.

b) Detailed description of every participating element, including its characteristics of tank and armament.

c) The tanks in the memory are 'moved' taking into account the factors that govern movement like terrain, obstacles etc. Gains and losses are decided upon, appropriate to firepower, accuracies of the gun, rate of fire, penetration etc.

d) Factors deciding visibility such as cover, concealment and direct visibility are catered for.

e) Tactical decisions are made by the users. They type in command and querries and receive replies in a normal form.

f) A set of factors additional to that mentioned above like artillery, anti-tank weapons,air support could be included to give more realistic and complex view.

To complete the simulation of a squadron level tank exercise, in addition to this part, development of the requisite interface to couple the necessary consoles and other aids used is required. This could be done as a further development to this thesis. When this is done, then this could be used to exercise the squadrons i.e. the squadron commander along with troop commanders for a tank battle.

1.4 Chapter Summaries:

In the next chapter we review the computer aids available for tactical exercises and war gaming. In this methods available for war gaming, the importance of Tactics/Strategy and any known work in this field will be discussed.

The complete 'over view' of the system will be given in Chapter III. This will give the block diagram of the system developed and what is expected of each block. Interconnections between these blocks and how to use the developed system will be described.

Chapter IV will give some specific algorithms that have been used. Approximations and assumptions made in developing this system will be discussed with possible improvements and the type of computer required for implementation of such a programme.

The graphic aid used by the simulator and a 'test exercise' will be described in the next chapter. The last chapter gives the conclusions and suggestions for further work.

CHAPTER II

WAR GAMING ON COMPUTER

War gaming has become a recognised technique in relatively recent times,but its essense has been a part of the pre-planning that military commanders have done through the ages. The military services would define a war game as a simulation, by whatever means, of a military operation involving two or more opposing forces, conducted, using rules, data and procedures designed to depict an actual or assumed real life situation. The simulated warfare provides a means of gaining experience, identifying errors or short-comings and improving skills without paying the penalties of the real war. Any thing in this world which is as costly as a war impels a search for its substitute.

2.1 War Gaming:

In military most of the important problems are amenable to analytical study. At present war gaming is mainly used for the following purposes:

a) To train military personnel

b) To test plans of commanders

c) For research, to explore new concepts.

Generally the above three purposes are clearly specified and well defined for each game. In some cases it could be used for multiple purpose as well, like for training as well as testing of plans. Games for research are usually conducted to study matters associated with new concepts. They also include the evaluation of plan and/or the formulation of doctrine in the application of new concepts. These three major purposes of war gaming are very much inter-related and also represent the stages of progressive development of war gaming.

War games normally attempt to develop and use models to represent the actions in a game. The model is important in a simulation. In a war game some factors not explicitly quantified in the model can be added by the controller, specified in the rules as input data, or covered as rules in the game assumptions. So effective models of battle can be built for some of the important factors involved in combat, but not all. Too little is known about the geometry of battle. Even information on the rate of casualty production and the resulting effects is inadequate to build precise mathematical equations. Formulae and model from historical records have some validity but have predictive value in a general way only. Although such models are

useful, they do not provide all the interactions and measures of effect needed for many analytical purposes. The object of the game will then be served by comparative rather than absolute casualty figures. Such a model is only as good as the capability of the model builder to build military sophistication and battle logic-expertise into the model.

2.2 Tactics/Strategy:

Some basic characteristics of war games normally found in all of them are:

a) Every war games simulates a military operation (irrespective of phase or manner of gaming).

b) Each game involves two or more opposing forces.

c) Each war game is conducted in accordance with data, rules and procedures acceptable to the military profession.

d) Every war game represents an actual or assumed real life situation.

An activity cannot be termed a war game unless military forces are involved in movements or operations accompanied by the clash of arms or the threat of such a clash. War

games are played only under conditions of simulated war-fare i.e. shooting, attack, or invasion by one force upon the territory of another. War game rarely consider non-shooting situations unless the situation is an interlude during which forces get into position or range to fight.

Therefore a war game is built upon a situation such as exists or realistically could exist in some locale under certain assumed conditions. The situation is described in detail sufficient for the commander to visualise the conditions under which he must conduct military operation.

Taking the above conditions in view the following ingredients are necessary for war gaming:

a) Knowledgeable personnel
b) Special facilities and equipments
c) Standard data, rules and procedures
d) A description of the situation to be represented in the game.

2.3 Other work in this Area:

The 'Tin Soldier' which became operable in 1952, is believed to be the first war game designed as a mathematical model for use on a computer in analytical research studies.

This war game is simple and is described as follows. A map is given for the area of interest and on it are

marked the various status of troops of pre-battle conditions for both the opposing forces. It essentially involves two forces, own and enemy forces. Then one man each is permitted to 'head' these forces. Their aim being to guide their forces in an imaginary battle. The battle is then organised to progress under the rules agreed as follows.

a) Each man is permitted to move his tanks/forces that would correspond to 't' seconds elapsed time on real terrain, consistent with the mobility capabilities of the tank.

b) After each man moves his tanks/troops according to his military good sense, tanks within range are assumed to bring fire on the opposition. The winner of tank duel is decided by flipping a loaded coin that is supposed to express the odd on the battle outcome in actual fighting between these tanks.

c) The battle is over as soon as all tanks on either side have been knocked out or have made successful withdrawal from the battle field.

The next step to this was the computerised Monte Carlo version called 'The maximum Complexity Computer Battle', capable of dealing with more of the variables of actual battle. The technique employed is called 'pseudo-random

number generator'. Every time this random number generator is referred, a large sequence of numbers (R) are computed, so that the sequence is practically random as far as the observer is concerned. If the probability of an event 'E' taking place is 'p', p being between '0' to '1', the following procedure simulates the occurrance of E:

if $R < p$, E occurs otherwise not.

Every time procedure is called, R is re-computed and compared with 'p'. Over a large number of calls, 'E' will occur the specified percentage of times. In addition in deciding the occurrence or non-occurrence of probabilistic events, the Monte-Carlo technique also provides for the determination of variables used in simulation in a probabilistic manner.

While the technique described here is satisfactory as a first approximation, it is necessary to add several refinements in practice. The uniform distribution of random variables is not common in practice.

Other methods used for formulation are the 'Next-Event formulation' and 'Fixed time step formulation'. The basic difference between the two is of time of computation. In the first i.e. next event formulation the time period used

is the time that will elapse between two subsequent events. Hence this time period varies as the simulation progresses. Whereas in 'fixed time', the time period of computation is kept constant to a pre-determined value.

With the 'next event' model, the processing of event occurences proceed occurence by occurence as follows. The simulation time is advanced to the time of the earliest future occurence. That occurence is processed, then the simulation time is advanced to the time of the (new) earliest occurence and that occurence is processed and so on. However, with the fixed time-step model, the processing of event occurences does not proceed occurence by occurence in this way. Rather it proceeds by group or batches of event occurences. All event occurences during the time step are processed together.

With the 'next-event' model the need to choose an arbitrary and artificial increment does not exist, and the pit-fall of picking a suitable increment is avoided entirely. In this respect, the next event formulation is preferable to the fixed time step formulations. Further more in a next-event model two event occurences are not processed as simultaneous events unless they bear identical occurence times. In a time-step model, on the other hand, batches of

non-simultaneous occurences are grouped together and made to appear simultaneous. Thus, simultaneity is created where it need not exist. In order to process simultaneous occurences,any simulation model must make use of some rule for determining the sequence in which to process the occurences. Fixed time-step simulation then, often takes a group of non-simultaneous event occurences, forces them to be simultaneous and process them in some sequence.

One situation, however, may require a user to use a time-step model instead of a next-event model. This is the situation where a next-event model takes a prohibitively long time to run on a computer and where the artificialities and inaccuracies of the time step model do not destroy the usefulness of the results. If the time increment, in fixed time step is sufficiently large, it is possible that the fixed time-step simulation may progress significantly faster than the corresponding next event simulation.

In this thesis the model of fixed time-step which can be modified to different values during the simulation is used. The capacity to modify the time step gives the option to the user to make a trade off between accuracy and speed. Thus the user is able to decide during the

simulation process, depending on the status or the situation that exists at that moment. If the user feels that the time step is to be reduced or increased, he can do so at that time.

2.4 Inter-active Simulation:

On computer there are two main types of simulation, one is complete computer simulation and another is man-computer simulation or interactive simulation. In the all computer simulation the input data and the rules are pre-described and given to the computer. The computer then simulates that action and reaches the result or conclusion. In this a number of runs would yield similar results because of the uniform strategy/decision of processing. Whereas inter-active simulation is different and would give different results with each user.

In this ,instead of producing a complete history automatically in one continuous operation such a routine processes events until an event is reached where a human decision is necessary or additional input is required. Then the routine causes the computer to wait for appropriate action to be taken before it starts another 'computer phase' of simulation. This is near to real time exercises

conducted by the armed forces and therefore better suited to simulate battle exercises. The simulator described here uses this interactive process of simulation.

Each interval during which the computer waits for input or for decisions is called a 'wait phase' of simulation. During this phase one or more team is in action. A team may be deciding on the movements of units, determining firing and communication policies to be followed, assessing information they have received about their own and enemy's units, and so on. The end of this phase occurs when the information requested by the computer is made available to restart the computer simulation routine.

The most obvious difference between a computer routine for man-computer gaming and a complete computer simulation routine is that the former must be able to 'converse' with the human players. For instance, the routine must be able to inform the players whenever any unit require repair, fuel etc. The routine must also inform the human players of enemy detections, firing of weapons, and damage sustained or inflicted. Each message is delivered through a computer output unit to the appropriate team or player.

An interactive game has some undesirable features which are associated with the alternating of the game. For example,

a 'wait phase' once begun must continue until it produces the decisions or other information which allow another computer phase to be started. Since this corresponds to stopping battles while decisions are being made, it is a poor representation of a real situation. Any attempt to place time limits on the 'wait phase' of the game requires rules which bear little relation to the real situation that one is trying to portray. Also the computer phase of the game must stop whenever any team must furnish information required by the computer simulation routine. This may cause delays unexplainable to one team or the other. For such reasons as these, use of 'synchronized' man-computer game is used where both 'wait' and 'computer' phases run continuously and concurrently with the use of 'clock'.

CHAPTER III

OVER VIEW

Tank battle simulation consists of essentially the storage of data concerning each tank and its subsequent updating depending on the commands given by the user. The data on each tank is stored in a record TNKDAT and has the following fields.

TYPNO: (1..5); Gives the type of Tank. Caters for a maximum of 5 types.

MAPCORD: X,Y; (INTEGER); Gives location of the tank in X and Y co-ordinates.

ORNTASN: (0..360); Gives the direction in which the tank is facing.

AMNCAP :TYP1, TYP2, TYP3: (INTEGER); Gives the amount of ammunition that is available with the tank. Caters for 3 different type of ammunition.

FUELCAP: (INTEGER); Gives the Quantity of fuel that is available with tank.

ALT : (INTEGER); Altitude in metres where the tank is located.

LODTME: (INTEGER); The time taken to load the shells before it is fired once the command to fire has been given.

STATUS:

BOGGED:(BOOLEAN); States whether the tank is bogged.

MOVING: (BOOLEAN); Whether the tank is moving, used to calculate rate of fire etc.

CANMOV:(BOOLEAN); Used when command to move is given.

FIRING:(BOOLEAN);States whether the tank is firing on target.

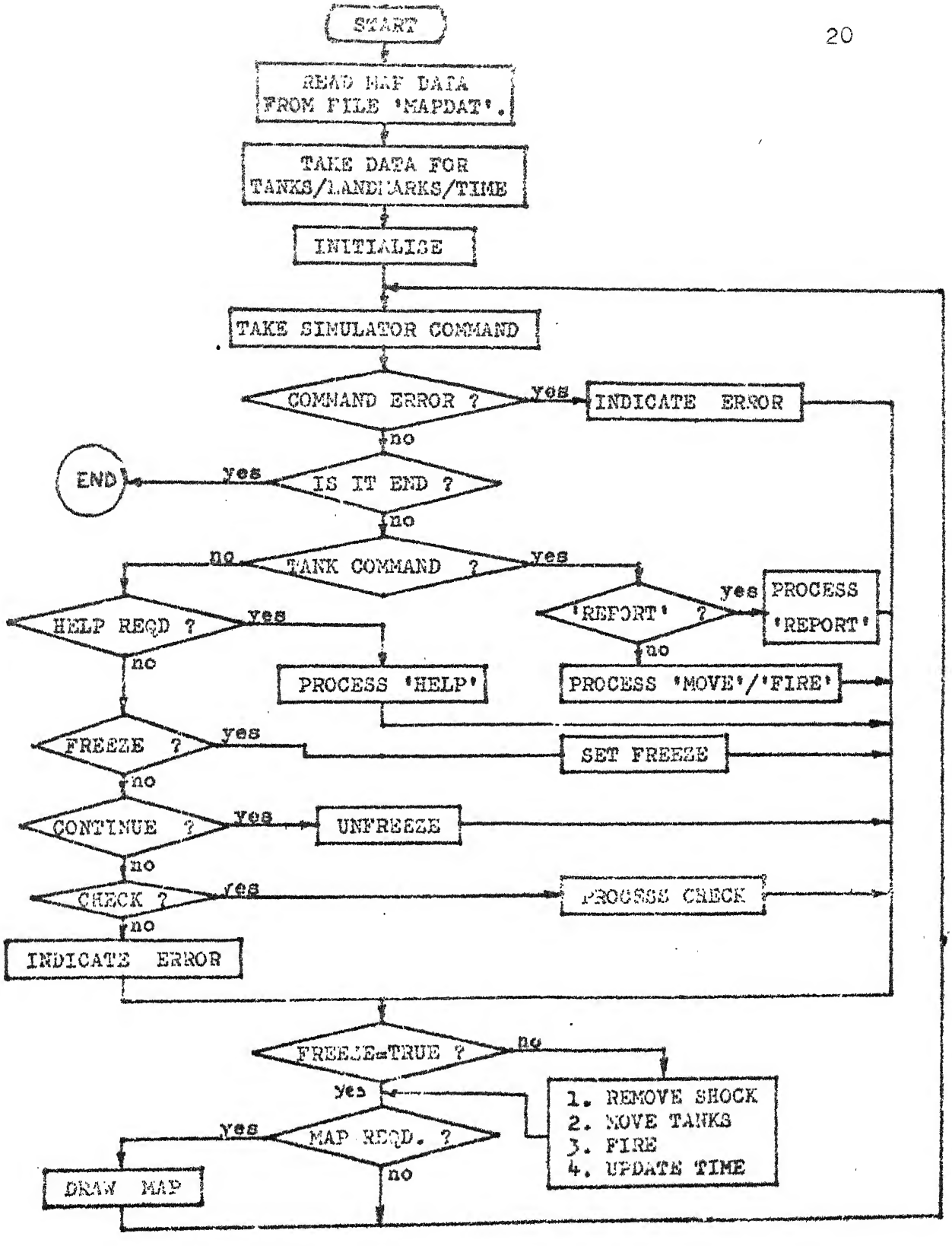
CUPOLACLOSED:(BOOLEAN); Used for visibility calculation.

NUETRALISED:(BOOLEAN); Whether tank is incapacitated and can take no further part.

SHOCKED:(BOOLEAN); Temporarily out of action, will not respond in that duration.

Various data from these fields can be 'read' through REPORT procedures which help the user in taking decision. These data also form the base for generation of other tables.

The simulation of MOVE and FIRE are done through the use of temporary tables which are processed to change the records of the affected tank in TNKDAT.


START
READ MAP DATA FROM FILE 'MAPDAT'.
TAKE DATA FOR TANKS/LANDMARKS/TIME
INITIALISE
TAKE SIMULATOR COMMAND
COMMAND ERROR ?
yes
INDICATE ERROR
no
END
yes
IS IT END ?
no
no
TANK COMMAND ?
yes
HELP REQD ?
yes
no
'REPORT' ?
yes
PROCESS 'REPORT'
no
PROCESS 'HELP'
PROCESS 'MOVE'/'FIRE'
FREEZE ?
yes
SET FREEZE
no
CONTINUE ?
yes
UNFREEZE
no
CHECK ?
yes
PROCESS CHECK
no
INDICATE ERROR
FREEZE=TRUE ?
no
yes
1. REMOVE SHOCK
2. MOVE TANKS
3. FIRE
4. UPDATE TIME
yes
MAP REQD. ?
no
DRAW MAP

The flow chart for the simulation process is shown as Fig. 1. This could be divided into the following sub-systems.

a) Map Generation

b) Initialization

c) Command Processing

d) Report

e) Information/Help

f) Update Records

g) Action

h) Graphic Display

3.1 Goals for Each Block:

Map Generation:

This part deals with the description of map details. This is a time consuming process and hence is separated from the main programme. The map is drawn with an interactive dialogue. The validity of the input with respect to the restrictions imposed for drawing the map are checked and then a file is created for the use of the main programme. This consists of the following 2 parts.

(a) Checking of Input:

Due to the limitations and economy of computer usage

some restrictions have been imposed for describing the details of a map. They are

i) Grid points are limited to a maximum of 40x40.

ii) Each grid point is at a distance of 500 mtr apart.

iii) Details such as rivers, canals etc. have been approximated as straight lines between turning points.

(b) Creating Data Files:

The above facts are checked as and when the map is being drawn. If it is within these constraints then this data is written into a file 'MAPDAT' which forms an input to the main programme. In addition another file 'MAP' is created to facilitate the drawing of map on Graphic terminal and avoid duplicating of calculation.

Initialise:

This is used for starting the simulation of the exercise. This consists of initialising the following:

(a) Map Details:

Reading the map data from file MAPDAT and storing the map details.

(b) Tank Details:

Initialising/reading the characteristics of the type of tanks used and their location at start.

(c) Graphics:

Initialising the graphic device to be used and resetting its data file MAP for its use.

(d) Land marks:

Taking in interactively the various landmarks that are likely to be used during the exercise and storing them for reference in LMTAB.

(e) Time:

If required, updating the 'Time step' to be used for each iteration. Also takes in the start time of the exercise.

(f) Variables:

Initializing the various pointers arrays and variables used in the programme.

Command Processing:

This is the main block of the system. It takes in one command at a time, analysis it and then appropriately calls the block required. The following facilities are available:

a) The command is analysed for its syntax error and if error exists, then this is indicated to the user.

b) Facility to 'freeze' all action and 'un-freeze' it when required.

c) Updates the time that has elapsed since the start time, if action is not frozen.

d) Process 'help' for MOVE, FIRE and REPORT.

e) Change time step if required, during execution of the programme.

Report:

This block is essentially a report back answer to querries addressed to the various tanks. Most of the details are taken from record TNKDAT. The report could be concerning the following.

(a) Location :

This reports the 4 figure grid reference on the map. This could be of

i) Self

ii) Another tank.

(b) Status:

This concerns the various activities that a tank could do or is doing. This refers to whether the tank is

i) Bogged
ii) Moving
iii) Can move
iv) Can shoot
v) Nuetralized

Once a tank is nuetralized then it can take no further part in the exercise and will not respond to any command.

(c) Visibility:

Checks if the other tank or location is visible to the addressed tank taking into consideration the following.

i) Time of day
ii) Equipment available for visibility at that time of day.
iii) Cover available to target
iv) Intermediate obstruction in the line of sight of these 2 points.

(d) Balance:

Balance of 2 commodities can be asked for

i) Fuel: This includes reserve as well

ii) Ammunition: This can be asked for a specific type of ammunition i.e.

(aa) HE (High Explosive)

(ab) AP (Armour Piercing)

(ac) SMOKE

By default, if no particular type is specified, then all types will be listed with their balance.

Information/Help:

This contains the information regarding the various modules, facilities that are available and how to use them. It essentially contains information regarding the following:

(a) HELP IN

i) Commands

ii) Report

iii) Move

iv) Fire

(b) CHECKING

i) List of land marks defined

ii) List of Tanks on move

iii) To change time step

iv) Set/unset drawing of a map

v) Display map.

This can be called for by HELP/CHECK and then choose the information in the type required by you. For 'help' it lists out the possible commands in that field with options. This forms the command communication language and is approximated to the actual commands that are normally used, keeping in mind its application on the computer. Commands available are given as Appendix A.

UPDATE RECORDS

When ever any 'Move' or 'Fire' command is given, details regarding the affected tanks is stored in records for further processing in the 'Action' part. This part does the following.

(a) Checks if the command is correct otherwise the error is indicated to the user.

(b) If the command is correct then it extracts information to be stored in the respective Move/Fire tables. The details are:

i) For Move: This data is stored in a chained record-MOVTAB with following details:

(aa) Starting location

(ab) Destination

(ac) Route - maximum of 10 intermediate points are catered for.

(ad) Current location

(ae) Name of tank

ii) For FIRE: The data for this is stored in record-FIRETAB . It consists of the following details

(aa) Name of tank

(ab) Target location

(ac) Target tank No. if any

(ad) No. of shells to be fired.

(ae) Rate of fire

(af) Hit probability

(ag) Type of shells to be fired

These tables form the basis for the other blocks which do the 'Action' part. Once the action is complete then that particular record is deleted from the chain. Therefore only the currently required information is stored in these chains.

Action:

This is essentially simulating the 'move' and 'fire' action in the system. The input to this is taken from the records FIRETAB for 'Fire' and MOVTAB for 'move'. For every cycle of time the following action part is done.

(a) For MOVE: For each of its record in MOVTAB it does the following:

i) Checks if the status is not nuetralized, if it is not then it will start the move process.

ii) It will then take its next objective and calculate the next grid point that it has to move.

iii) Before 'moving' to the next grid point, checks for its topography whether there is a river, canal, minefield etc. and takes appropriate action.

iv) If it can move to the next grid point then calculates the following

(aa) Time taken for this move

(ab) Fuel spent

and up dates both these quantities.

v) Thus it will continue till either it reaches its destination, is stopped or the time step is over.

(b) Fire: When 'fire' is to be done the following is done for each record of its table 'FIRETAB'.

If the status of the tank is not nuetralized then each time period, it will fire the shells depending on the following

i) Rate of fire of the tank depending on whether it is moving or stationary. This is taken from the tank status in TNKDAT.

ii) Calculate the location of hit taking into consideration the probability of hit and ranging.

iii) Analysing the hit with respect to the target tank i.e. angle of hit, distance and type of ammunition used. Then accordingly changes the status of the effected tank if needed.

iv) It will continue this till the required number of shells are fired or time step finishes or is commanded to stop firing.

Graphic Display:

Then lastly the complete activity is displayed on the Graphic terminal. The display consists of the following

(a) Details of the map which include the following

i) Rivers

ii) Roads

iii) Canals

iv) Minefields

v) Marshy ground

vi) Fields/High grass

(b) The map displays the area of 20x20 km divided into 40x40 grid points i.e. each grid point is at a distance of 500 mtr apart.

(c) For referencing on the map, grid marks are available at 2.5 kms distance (5 grid points).

(d) The 'land marks' defined by the user is also displayed at the location with the assigned name.

(e) Location of all tanks, as they are located at the time of display of the map.

(f) Location of hit of the shell at the time of display if there is any.

(g) Contours are displayed at an interval of 50 mtr. This is done with relative height, with the surrounding ground taken as '0'.

3.2 Interconnections:

The common record for the complete programme is TNKDAT. There are other subsidary records to this record which are used to update this during the exercise.

The map data generated is primarily used for the 'move' and display purposes. This data is kept in GRIDCHAR. When the move is simulated then details regarding the ground is used for calculation of speed, fuel spent and obstacles enroute.

After calculation of fuel spent and new location, they are updated in the record TNKDAT. If there is any change in status like the tank is bogged, nuetralized etc.

the same is changed in the 'status' field of TNKDAT. Therefore the latest information is stored in the record TNKDAT.

When 'Fire' has to be done. The shells are fired from the tank depending on the command given and the same number as and when fired are reduced from its record of that particular tank. The location of hit is calculated and then the effect of shell analysed depending on the type of shell and location of target damage if any is reflected through TNKDAT.

'Shocked' is a temporary status wherein the tank is not able to respond to your command for one cycle of time. The 'shocked' status removal is directly done through the main programme.

3.3 How to use the Simulator:

There are 2 programmes in the system which are to be used in the following sequence.

(a) MAPRIT.PAS

(b) BATTLE.PAS

MAPRIT.PAS

The programme MAPRIT.PAS is used to describe the topography of the area that is being used. The following

data regarding the details to be described must be available at the time of execution.

(a) RIVERS/CANALS

i) Direction of flow of water

ii) Depth of water in cms

iii) Width in mtrs.

(b) OTHERS: For all others including the rivers and canals the following are required.

i) Start point

ii) Turning points

(c) CONTOURS: The height of the contour, in steps of 50 mtr, relative height with the surrounding area taken at '0'.

All the start points and turning points are to be given as 4 figure grid reference. In between these 'Turning Points' the particular detail is approximated as a straight line, hence the points have to be chosen accordingly. While describing the details restriction is also made for the direction in which the next grid point exist. The next grid point has to be in one of the 8 cardinal direction i.e. N/NE/E/SE/S/SW/W/NW.

When the next grid point has been wrongly directed, a message is given i.e. 'going out of map area - direction

wrong'. This point could be recalculated and given.

All contours must be complete i.e. open contours might lead to wrong result hence not allowed.

Any number of such details can be described but each one has to be described separately. For example a river and its tributary will be described separately.

Once this programme is executed two files 'MAPDAT' and 'MAP' are made. They can be used for any subsequent exercise till another map, is defined in a similar manner.

BATTLE.PAS: The execution of this programme will simulate the tank battle. The following data is to be kept ready at the start of the exercise and given when asked.

(a) Type of tank used: Each tank has particular data associated with it for its armour and guns. The following 2 tanks have been catered for:

i) Vijayanta

ii) T-55

(b) Location of Troops: Location of 5 troops of a squadron of tanks in 4 figure grid reference.

(c) Time: The time of start of simulation of the exercise is to be given. Along with it the time

step to be used for the exercise is to be given. The minimum time step is calculated as 1 min. This time step can be changed during the execution of the programme as well.

(d) Land marks: The land marks that are likely to be used during the exercise with their name (less than 12 chars without blanks) and its 4 figure grid reference. Any number of such land marks can be defined.

Once these details have been given the simulation of the exercise starts. There are various commands that can be given. There are basically 2 types of commands i.e. one that is addressed to a tank or otherwise.

The commands that could be addressed to a tank could be

a) To report about centain attributes

b) To move from one location to another

c) To fire at a location or tank.

These are given as Appx. 'A'. The other commands that could be given to the simulator are

(a) Help: To find the information regarding the commands addressable to the tanks.

(b) Check: The checking/changing of various status during the exercise like

i) Land marks

ii) Time step change

iii) List of tanks on the move

iv) Setting/Unsetting of map drawing.

(c) Freeze/Cont: This will freeze/unfreeze all the action part. This could be used at the time when some details have to be studied without change in status. 'Cont' will unfreeze the action.

(d) End: Used to terminate the programme.

Generally the following guidelines are to be used during interactive communication with the simulator.

(a) All sentences or where possibility of a sentence exists, the reply has to end with a '.'.

(b) A word is taken as 12 characters without blanks.

(c) The delimiters between words are 'blank/./,.

(d) All single word reply need not end with a '.'.

(e) For a Yes/No type of diaglogue, a simple 'Y' or 'N' could also be used.

The tank addressing is done by 'NO'. The 'NO' consists of 2 parts, Troop number followed by Tank number. Say there

are 5 troops and each troop having 3 tanks then TK 31 will mean Tank No. 1 of troop No. 3.

Normally map drawing will not be available. If this is required, then this will have to be set through CHECK. A one time map drawing is also available through this procedure.

CHAPTER IV

ALGORITHMS

In this chapter some algorithms will be discussed along with the assumptions/approximations that have been made.

4.1 Fire:

The procedure FIRE simulates the 'fire' part. The list of tanks that have been ordered to fire and details required is kept in a linked list of records FIRETAB. In each time step the number of shells that are to be fired is calculated, their effect analysed and reflected in the tank status of the affected tank. Once all the 'ordered' shells have been 'fired' by that tank then that record is deleted from the list.

The various fields in the record FIRETAB are as follows:

SOURCE TK:(INTEGER); Used for storing the number of the tank that is firing.

DEST TK :(INTEGER); Used to stores the number of the tank that is being fired upon. If it is '0' it indicates that firing is on a location.

SOURCE LOC: X,Y:(INTEGER): Stores the location co-ordinates of the tank firing the shells. Used for calculating distance and hit.

DEST LOC: X,Y:(INTEGER); Stores the location of tank being fired upon. Used for hit analysis and 'aiming'.

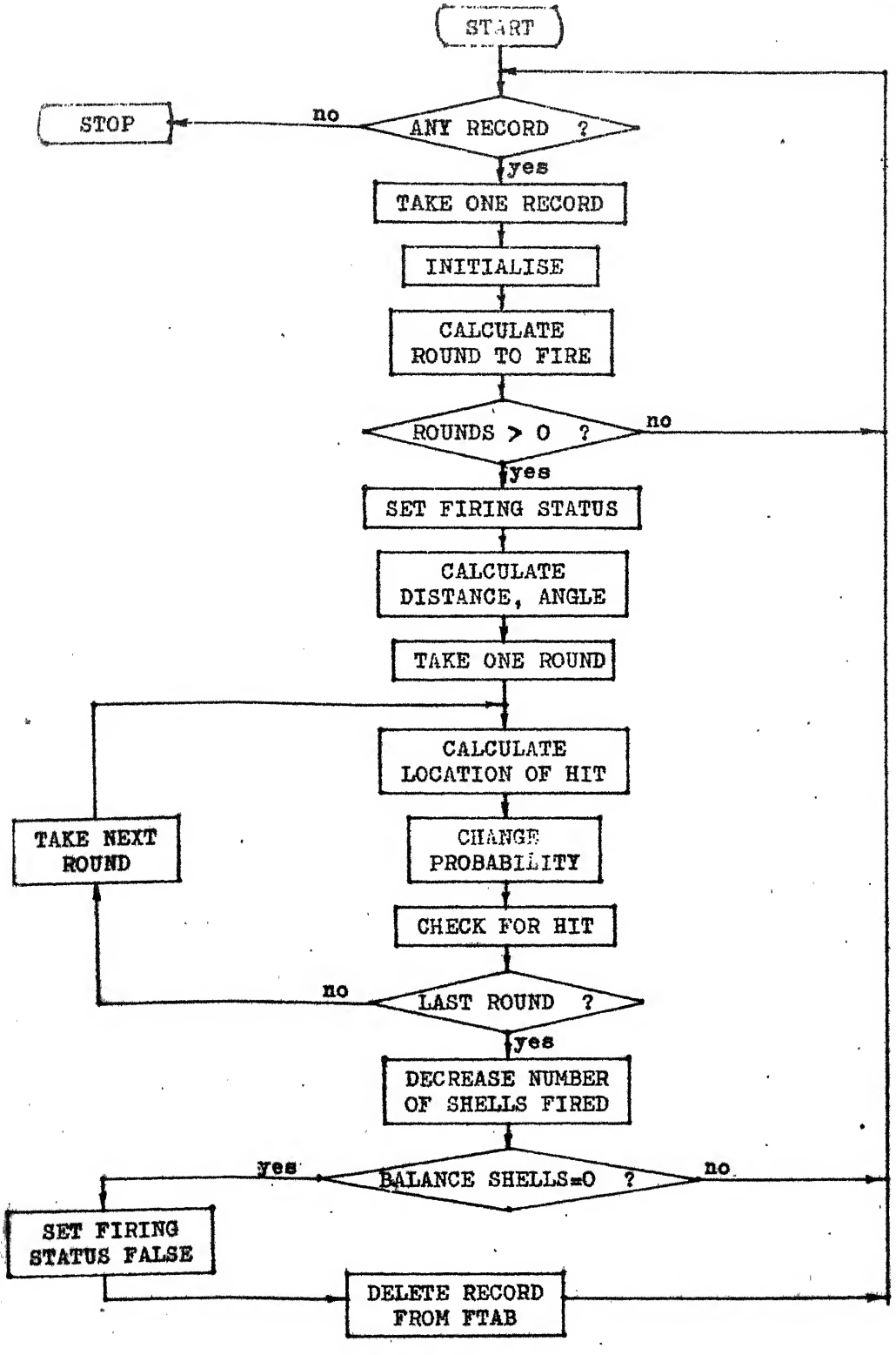
TOTNUM: (INTEGER); Gives the total number of shells that are left to be fired.

TIME: (INTEGER) ; This gives the load time i.e. time taken by the tank to initially start its first fire.

HIT PROB: (REAL) ; Gives the hit probability of the shell. Used for calculating the location of hit of the shell.

RATE : (0..15); This gives the number of shells that can be fired in a minute. This depends on the status of tank like moving etc.

ULINK, LLINK:(FPTR); Used to form a linked list of all the tanks that are firing and random deletion on completion of 'fire'.


START
STOP
no
ANY RECORD ?
yes
TAKE ONE RECORD
INITIALISE
CALCULATE ROUND TO FIRE
ROUNDS > 0 ?
no
yes
SET FIRING STATUS
CALCULATE DISTANCE, ANGLE
TAKE ONE ROUND
CALCULATE LOCATION OF HIT
TAKE NEXT ROUND
CHANGE PROBABILITY
CHECK FOR HIT
no
LAST ROUND ?
yes
DECREASE NUMBER OF SHELLS FIRED
yes
BALANCE SHELLS=0 ?
no
SET FIRING STATUS FALSE
DELETE RECORD FROM FTAB

F I R E

ALGORITHMS

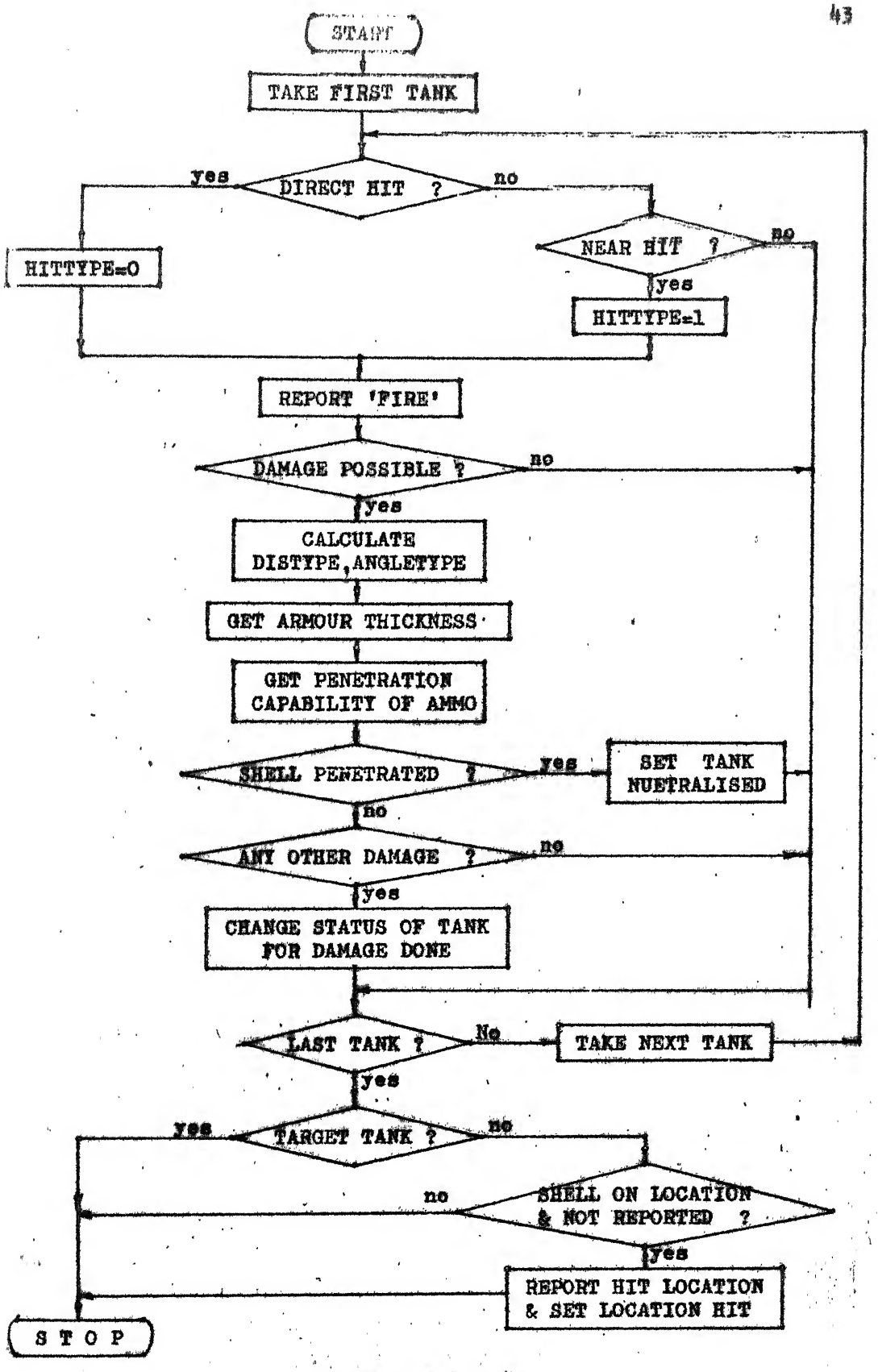
FIRE

```
1. FOR EACH RECORD IN LIST FTAB DO
   BEGIN
   2. INITIALISE
   3. CALCULATE NUMBER OF ROUNDS TO BE FIRED IN THIS CYCLE
   4. IF NUMBER OF ROUNDS GREATER THAN ZERO THEN
      BEGIN
      5. SET STATUS OF THAT TANK TO FIRING
      6. CALCULATE DISTANCE & ANGLE TO TARGET
      7. FOR EACH ROUND DO
         BEGIN
         8. CALCULATE LOCATION OF HIT
         9. CHANGE PROBABILITY OF HIT
         10. CHECK HIT AND ANALYSE ITS EFFECT
         END
      11. DECREASE NUMBER OF SHELLS FROM BALANCE
         12. IF BALANCE OF SHELLS TO BE FIRED EQUALS ZERO THEN
         BEGIN
         13. SET FIRING STATUS TO FALSE
         14. DELETE THIS RECORD FROM FIRETAB
         END
      END
END
```

CHECK HIT: Checks if any of the tanks have been hit directly or indirectly. If it is hit then calls for analysis for damage. Also checks, if the target is a location, then if it has been hit or not.

```
1. FOR EACH TANK DO
   BEGIN
   2. IF HIT LOCATION = TANK LOCATION THEN
      ANALYSE HIT FOR DIRECT HIT
   3. IF HIT LOCATION NEXT TO TANK LOCATION
      THEN ANALYSE HIT  FOR INDIRECT HIT
   END
4. IF TARGET IS NOT TANK THEN
   BEGIN
   5. IF HIT LOCATION = TARGET LOCATION AND
      HAS NOT BEEN REPORTED THEN
      BEGIN
      6. REPORT LOCATION HIT
      7. SET THIS TO REPORTED
      END
   END
```

ANALYSE HIT: Reports target tank coming under fire and then analyses the hit for any damage it may have done and changes the status if required.


START
TAKE FIRST TANK
DIRECT HIT ?
yes
no
HITTYPE=0
NEAR HIT ?
no
yes
HITTYPE=1
REPORT 'FIRE'
DAMAGE POSSIBLE ?
no
yes
CALCULATE DISTYPE,ANGLETYPE
GET ARMOUR THICKNESS
GET PENETRATION CAPABILITY OF AMMO
SHELL PENETRATED ?
yes
SET TANK NUETRALISED
no
ANY OTHER DAMAGE ?
no
yes
CHANGE STATUS OF TANK FOR DAMAGE DONE
LAST TANK ?
No
TAKE NEXT TANK
yes
TARGET TANK ?
yes
no
SHELL ON LOCATION & NOT REPORTED ?
no
yes
REPORT HIT LOCATION & SET LOCATION HIT
STOP

C H E C K H I T

1. IF NOT REPORTED THEN REPORT COMING UNDER FIRE
2. IF AMMUNITION IS NOT 'SMOKE' OR DISTANCE WITHIN RANGE

   BEGIN

   3. CALCULATE DIST-TYPE & ANGLE OF HIT ON THE TANK
   4. GET ARMOUR THICKNESS AT THAT POINT
   5. IF THICKNESS< PENETRATING CAPABILITY

      THEN SET TANK NUETRALISED

      ELSE

   6. ANALYSE FOR EXTENT OF DAMAGE TO TANK AND CHANGE

      STATUS ACCORDINGLY

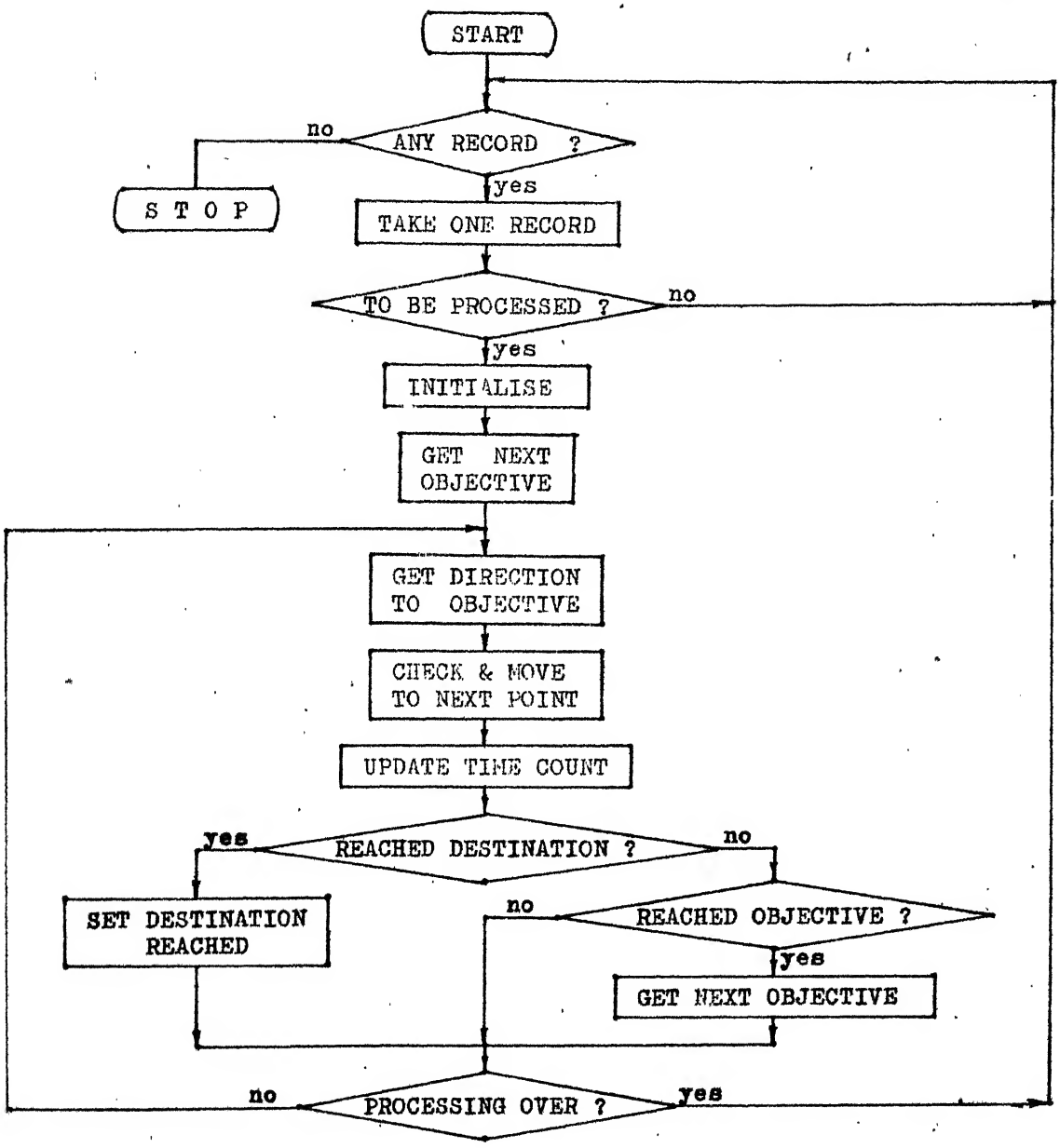
   END

4.2 MOVE TANKS:

This procedure simulates the 'move' part. The details regarding all the tanks that have been ordered to 'move' are kept in a linked list of records in MOVTAB.

In each cycle, each tank in this list is moved for a time period of one time step unless it comes to a halt in between. During the move it 'jumps' from one grid point to another. Before going to the next point it checks for its topography and its possibility to move to that location.

If some obstacle is encountered then it takes action depending on the type of obstacle and command given. The change of status if any is reflected. Once the destination


START
ANY RECORD ?
no
yes
S T O P
TAKE ONE RECORD
TO BE PROCESSED ?
no
yes
INITIALISE
GET NEXT OBJECTIVE
GET DIRECTION TO OBJECTIVE
CHECK & MOVE TO NEXT POINT
UPDATE TIME COUNT
yes
REACHED DESTINATION ?
no
SET DESTINATION REACHED
no
REACHED OBJECTIVE ?
yes
GET NEXT OBJECTIVE
no
PROCESSING OVER ?
yes

M O V E T A N K S

is reached or has been nuetralized or incapacitated permanently then this record is deleted from the move list MOVTAB.. The various fields in the record MOVTAB are

TANK NO: (0..99); The number of tank on the move.

SOURCE : X,Y:(INTEGER); The location of the starting point. X and Y cordinates of the tank.

DEST: X,Y: (INTEGER); The location it has to reach ultimately.

LOC: X,Y: (INTEGER); The present location where the tank is located.

ROUTE:ARRAY(1..10) OF CORD:X,Y:(INTEGER); Gives location of the intermediate points.

STATE:(BOOLEAN); Whether the tank is moving or stopped temporarily.

OBJECTIVE:(INTEGER): The total No. of intermediate points.

COURROBJECTIVE:(INTEGER); Present No. of the objective.

ALGORITHMS

MOVE

1. FOR EACH RECORD IN LIST DO

   BEGIN

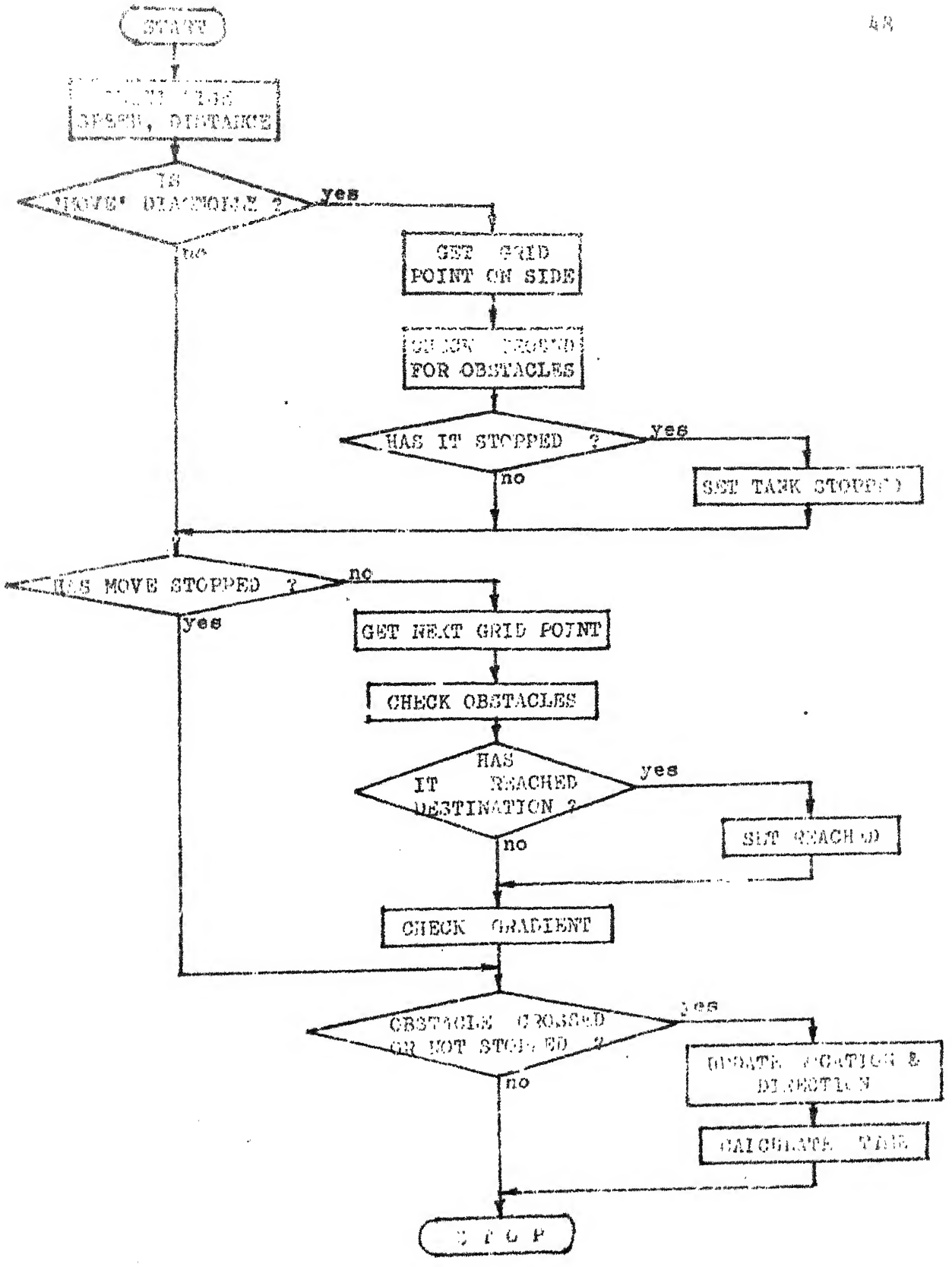
```
2. IF RECORD REQUIRES PROCESSING THEN
   BEGIN
   3. INITIALISE
   4. GET NEXT OBJECTIVE
      REPEAT
      5. GET DIRECTION TO OBJECTIVE
      6. COMPUTE TIME FOR MOVE TO NEXT POINT
         IF MOVE IS POSSIBLE
      7. INCREMENT TIME COUNT WITH CURRENT TIME
      8. IF REACHED DESTINATION,SET DESTINATION  REACHED
      9. ELSE REACHED OBJECTIVE THEN GET
         NEW OBJECTIVE
      UNTIL REACHED DESTINATION OR FINISHED
            MOVE OR TIME STEP OVER
   END
END
```

COMPUTE TIME

This procedure calculates the next point that the tank is to be moved. Then checks for the type of ground that is there between the 2 points and the next point. If it can move to the next point then the following is calculated.

a) Speed in this terrain

b) Distance to the point



COMPUTE TIME

c) Gradient between the points

d) Fuel consumed

If it is not possible to move directly then checks if it can encounter this obstacle taking into consideration the following

a) Type of obstacle

b) Capability of the tank

c) If water obstacle, then reports this and asks for orders. If ordered to cross then checks if it can, otherwise gets bogged.

COMPUTE TIME

1. INITIALISE DATA
2. IF TANK TRAVELLING DIAGONALLY THEN CHECK GROUND IN BETWEEN FOR OBSTACLES
3. IF MOVE NOT FINISHED THEN

BEGIN

4. GET NEXT GRID POINT
5. CHECK GROUND FOR OBSTACLES
6. IF GRADIENT WITHIN LIMIT THEN

BEGIN

7. CALCULATE GRADIENT
8. CALCULATE SPEED CHANGE

```
        9. FUEL CONSUMPTION CALCULATE
        END
    ELSE
    10. STOP TANK AT LOCATION
END
11. IF TANK HAS NOT STOPPED OR HAS CROSSED OBSTACLE
BEGIN
    12. MOVE TANK TO NEXT LOCATION
    13. UPDATE DIRECTION OF TANK
    14. CALCULATE TIME TAKEN TO REACH NEXT POINT
    15. CALCULATE FUEL CONSUMED AND
        DECREASE FROM BALANCE
END
```

4.3 Approximation/Assumptions:

For the simulation process some approximations and assumptions are made and these have been done keeping in view the realistic situation and its application on the computer. Some of these have been described in the succeeding paragraphs.

Distance between grid points is kept at 500 meters due to the following reasons:

(a) Depiction of a large area of 20x20 Kms used for the mobility of the tank.

(b) Decreasing the number of grid point for the same area thereby decreasing the memory requirements.

(c) Resolution in graphic display due to limitation of size of display which is fixed.

Straight line approximation between points. In this case error is maximum within 500 mtr length because any number of turning points can be described. The limitation in direction restricted to only 8 is a constraint mainly for storage of data and computation simplicity. The deviation to some extent is acceptable for the exercise purpose.

Only six details like river, canal, road etc. have been included for the map description. Other details which do not have much bearing on the tank battle like anti-personnel mines, barbed wire fence, booby traps etc. have not been considered.

The minimum time step has been kept as 1 minute taking into consideration the maximum time taken by a tank to move from one point to another in the worst possible condition.

The Move and Fire operations are done serially one followed by the other and not parallel as it happens in real time. This is due to computer programme control flow being sequential.

The 'shocked' tank is kept under shock only for one cycle of time but in actual practice this could be of variable length of time but is approximated as one cycle.

During the firing of shells, the rest of action is frozen as the time of flight is considered very small compared to movement during that period of time.

The height between 2 contours is kept constant throughout this area and height changes only from the next contour onwards.

4.4 Improvements:

In the 'MOVE' commands more general commands like move in formation, and deploy various tactical formation can be adopted and catered for.

Different type of firing tactics can be catered for. Here only 'Ranging' is used by SQUARE ROOT method.

More accurate analysis of hit of the shell could be done. In this, approximation is done upto an accuracy of ±250 meters which may not be accurate in certain cases.

More details in the map description could be given like tracks, rough or forested area or undulating ground as found in deserts which provide cover to tanks or obstruct visibility could be described and used.

To give more realistic view, Air threat, Artillery and Anti-tank weapons could be incorporated in the simulations.

A combination of MOVE & FIRE tactics used in the case of providing covering fire for an attack could be included.

4.5 Time & Memory Considerations:

The simulator programme is written in Pascal language and implemented on system Dec 1090 computer. All the timings and memory requirement are corresponding to this system.

(a) Programme Storage

BATTLE.PAS	116K bytes
MAPRIT.PAS	20K bytes

(b) Compilation data using

GPAS (Graphic Pascal)

Memory requirement = 105K bytes

Run time = 11 secs

(c) Programme Execution

i) Time for initialisation, including reading of map, tank locations, land marks Time etc. = 10 secs (average).

ii) Command Processing Times

Simulator command = 33 ms (average)

Tank commands = 16 ms (average)

iii) Action Processing time = 26 ms (average)

iv) Memory required during execution = 275K bytes (for programme example)

It must be noted that these timings and memory requirements will vary depending on the simulated exercise as dynamic storage and processing is used by the programme.

(d) Type of computer required: For the running of this programme any general purpose computer having the following facilities could be used.

i) Memory of 64K

ii) Terminals (TTY) for interactive usage

iii) Graphic Terminal device (Tectronix display terminal) for the use of displaying of map if required.

iv) Availability of Pascal graphics with its linking with GPGS.

CHAPTER V

GRAPHIC AIDS

The graphic aid used here for display purpose is Graphic Terminal (Tectronix display terminal) and the language used is Pascal (Graphics).

The size of display on this device is limited to physical size of 15 cms x 15 cms. On this is displayed an area of 20x20 kms divided into 40x40 grid points.

For easy referencing on the map '+' markers are displayed at an interval of 5 grid points i.e. distance of 2.5 kms. each.

The legend for the details represented with the markers that have been used are

a) RIVER (double line) =

b) CANAL (nabla) f

c) ROAD (straight line) -

d) MINEFIELD (cross) X

e) MARSHY GROUND (Diamond) ◇

f) GRASS/FIELD (Delta) △

g) CONTOURS (dot) .

The location of hit of a shell is displayed by a * at the location of hit. This is done only for those shells that

are fired in that cycle only. This is done by storing the hit location in a file FIREDT when the Fire procedure is executed. This data is used later by the programe DRAW MAP while drawing the map.

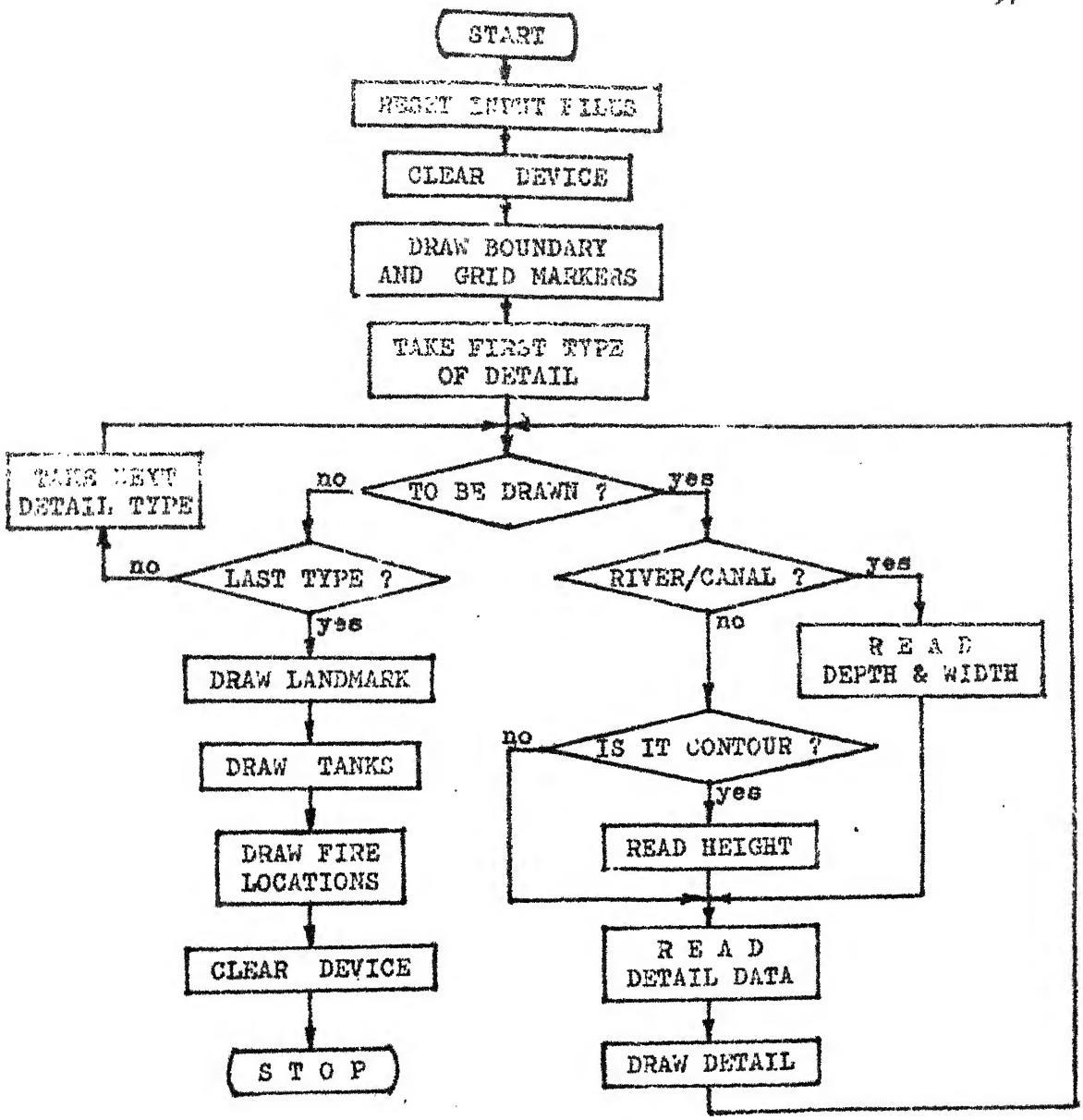
All land marks as described by the user is displayed on the terminal by the name at its location. Its exact location is given by starting character of the land mark name.

All tanks are displayed merely by their numbers i.e. Tank No. 3 of troop No. 1 is displayed as 13 on the displayed. The starting number gives the exact location of the tank.

The programme for Graphic Display is written in Pascal which does not at present provide all the facilities that are otherwise available in Fortran (GPGS). It is also handicapped by the lack of literature on it. Because of these restrictions graphic display is also restricted. This could be developed further as and when the Graphic Pascal becomes more powerful.

5.1 DRAW MAP:

This is the procedure used for drawing the map on the graphic terminal. It takes the input data from files MAPDAT, MAP and FIREDT. Apart from this the data for tanks location


START
RESET INPUT FILES
CLEAR DEVICE
DRAW BOUNDARY AND GRID MARKERS
TAKE FIRST TYPE OF DETAIL
TAKE NEXT DETAIL TYPE
no
TO BE DRAWN ?
yes
no
LAST TYPE ?
yes
RIVER/CANAL ?
yes
no
R E A D DEPTH & WIDTH
DRAW LANDMARK
DRAW TANKS
DRAW FIRE LOCATIONS
CLEAR DEVICE
S T O P
no
IS IT CONTOUR ?
yes
READ HEIGHT
R E A D DETAIL DATA
DRAW DETAIL

DRAW MAP

is taken from Record TNKDAT and for land marks from LMTAB. The flow chart for this is shown as Figure 6.

ALGORITHM

1. RESET INPUT FILES MAPDAT, MAP AND FIREDT
2. CLEAR GRAPHIC DEVICE
3. DRAW BOUNDARY AND GRID MARKERS
4. FOR EACH TYPE OF DETAIL DO
   BEGIN
   5. WHILE DETAILS OF THIS TYPE ARE TO BE DRAWN DO
      BEGIN
      6. IF DETAIL IS RIVER OR CANAL READ DEPTH & WIDTH
      7. ELSE IF DETAILS IS CONTOUR READ HEIGHT
      8. READ OTHER DATA REGARDING DETAIL
      9. DRAW DETAIL ON DEVICE
      END
   END
10. DRAW LANDMARKS
11. DRAW TANKS
12. DRAW FIRED SHELL LOCATIONS
13. CLEAR DEVICE IF SEEN

5.2 Test Exercise on Simulator:

For the purpose of demonstration, as to how this simulator could be used, a small test exercise is demonstrated

as follows. Assymptions made for this purpose and a narrative are given below. The aids used for this purpose are, the Hard Copy terminal for recording the interactive communication dialogues between the user and simulator and the 'plotter' to get the map output.

Exercise Background:

There are two states OWN DESH and ENEMY DESH, which have unfriendly relations. The NULLAH river which flows from loc 0127 to loc 3907 is the international boundary between them. Own Desh consists of troops 1,2 and 3 and Enemy Desh has troops 4 and 5. Their locations and various LANDMARKS are shown in MAP-1. Area HUT being a strategically important location Own Desh decides that it must capture it. It ventures to do this in the following manner.

(a) AIM: Troop 1 and Troop 2 to capture area HUT by 042300 Hrs.

(b) PLAN: The plan to capture HUT is to be done in 2 phases.

i) Phase 1: To cross river NULLAH and make a bridge, Troop 1 in area TREE and Troop 2 in area BRIDGE.

ii) Phase 2: Troop 2 to move and capture area NODE to give support to Troop 1 for the capture of area HUT.

(c) PROGRESS OF BATTLE: The course of events that take place during this operation are as follows:

i) Because of the increasing tension, Enemy Desh sends TK 53 to 0723 and TK 42 to 3117 for observation.

ii) To divert attention Troop 1 and 2 fire smoke shells at loc 0823 and 3113 to give the impression that a river crossing and attack is being planned in this area.

iii) Falling for this TK 51 and TK 41 are sent by Enemy Desh to locs 0822 and 3113.

iv) Troops 1 and 2 cross the river in area CROSS and BRIDGE respectively.

v) Troop 3 is brought up to Loc 1717 to give support to these troops.

iv) TK 52 moves to 1727 and TK 51 and TK 53 try and cross the NULLAH river to move from the back.

vii) Troop 2 moves towards NODE and after softening it with fire assaults on it to capture it.

viii) Troop 1 after nuetralising TK 52 moves to area HUT.

(d) The activity recorded on the map's correspond to starting of the exercise (MAP-1), after completion of phase 1 (MAP-2) and after completion of the exercise (MAP-3).

(e) The interactive dialogue used for this exercise is given as Appendix 'B'.

CHAPTER VI

CONCLUSION

A simulator for tactical exercise based on tank has been described here in reasonable detail. This simulator takes into account only a chosen set of factors and follows a specific set of rules built into it. Various constants such as those describing the accuracy of armament, the probability of muetralization and various kinds of hits are used by the simulator.

It is hoped that this simulator will form a logical extension of a sand model exercise, so that it provides more realistic details such that it gets close to real exercises in field. It will enable exercises in critical areas, such as enemy held territory or civilian territory in accessable for real exercises. By co-ordinating intelligence information with known topographical details of critical territories very effective familiarization with such areas can be ensured by simulator exercises.

The simulator goes beyond Armour battle and is very effective in training human decision making element in controlling complex operations, the outcome of which is primarily decided by equipment characteristics, fire power and tactics employed.

Man-machine communication is an important aspect which has been dealt with in this simulation. It is hoped that this will provide to the user how best to use this effectively to deal with a variety of command and control tasks.

It is considered that this simulator could be a useful training aid for Armoured Corp Officers in learning squadron level tactics. The simulator could be used both as a programmed teaching aid for minor tactics as well as training the squadron as a team.

6.1 Further Development:

A small operating system to simulate a squadron level battle to include the following features could be added.

a) 2 rival squadrons are able to operate independently.

b) Classification of messages, so that they are routed to various combination of leaders/tanks.

c) An umpire who gets all messages and commands given by all originators.

d) Partial display of the areas under their squadron to the Squadron commander and troop commanders.

e) An element of chance to intercept messages of the opposing forces.

REFERENCES

1. Capt. Gurmit Singh, 'War Gaming', M.Tech. Aug. 1979.

2. Maj. S.C. Gupta and S. Ramani, 'Computers and Tactical War Gaming', Technical Report No. 44, TIFR, June 1968.

3. Richard F. Barton, 'A Primer on Simulation and Gaming', Prentice-Hall Inc.

4. Melvin Dresher, 'Game of Strategy-Theory and Applications', Prentice-Hall, Inc.

5. George W. Evans, Graham F. Wallace, Georgia L. Sutherland, 'Simulation using Digital Computers,' Prentice-Hall Inc.

APPENDIX A

COMMANDS AVAILABLE WITH OPTIONS

COMMAND:: = TK (TANK NUMBER) (REPORT PART)/(FIRE PART)/(MOVE PART).

REPORT PART:: = REPORT LOCATION

STATUS TK (TANK NUMBER)/$ LOC (GRID REF)/$

ENEMY

BALANCE AMMUNATION/FUEL HE/SMOKE/AP/ALL/$

VISIBILITY TK(TANK NUMBER)/LOC (GRID REF)

FIRE PART:: = STOP FIRING

FIRE/SHOOT (NO) HE/AP/SMOKE SHELL(S) AT TK (TANK NUMBER)(DIRECTION)
LN (GRID REF)

MOVE PART:: = MOVE TO (LAND MARK) /$

ROUTE (LAND MARKS)

(DISTANCE) MTR/METER/KM (DIRECTION)

OF (LAND MARK)

ROUTE (LAND MARK)

NOTE:

TANK NUMBER :: = TROOP NO. TANK NO.
DIRECTION :: = CLOCK DIRECTION/CARDINAL DIRECTION/DEGREE
CARDINAL DIRECTION:: = NORTH/N, NORTH EAST/NE etc.
LAND MARK :: = DEFINED LAND MARK/GRID REF
LAND MARKS :: = LAND MARK, LAND MARKS/$
GRID REF :: = FOUR FIGURE GRID REFERENCE

GPGS-10 VERSION-001

```
THIS IS A PROGRAMME EXERCISE FOR TANK BATTLE AT SQUADRON LEVEL. YOU
ARE REQUIRED TO TYPE IN COMMAND IN THE FORMAT SPECIFIED AND IT WILL BE
EXECUTED. THE FOLLOWING POINTS NEED PARTICULAR ATTENTION:-
        1 THE COMMAND MUST BE TYPED STARTING WITH <TK> OR OTHER COMMAND
        2 THE COMMAND MUST END WITH <.>
        3 IN CASE OF HELP TYPE <HELPCOMMAND> OR <HELP>
        4 TO TERMINATE THE PROGRAMME TYPE <END.>
        5 AFTER YOU HAVE WRITTEN YOUR COMMAND GIVE CARRAIGE RETURN
        6 IF YOU WANT THESE COMMANDS TO TYPED THEN TYPE COMMAND <CHKPTS.
READY TO START NOW
PLEASE GIVE THE TYPE OF TANK USED -
if it is VIJAYANTA then type "V"
if it is T-55 then type "T"
type of tank :V
PLEASE GIVE THE LOCATION OF THE VARIOUS TROOPS OF YOUR SQUADRON
YOU ARE REQUIRED TO GIVE ONLY THE FOUR FIGURE GRID REFERENCE
TROOP NO  1:1519
TROOP NO  2:2511
TROOP NO  3:1511
TROOP NO  4:3133
TROOP NO  5:0531
IMPORTANT:note that the map details have been taken from your file ="MA


DO YOU WANT TO DEFINE LANDMARKS ?Y
TYPE IN LANDMARKS IN THE FOLLOWING FORMAT [<LM> <GR>.]
WHEN YOU HAVE FINISHED then type [FINISH.]
INPUT:HILL 1509.
INPUT:RDJN 2505.
INPUT:CROSS 1818.
INPUT:BRIDGE 2513.
INPUT:NULLAH 1021.
INPUT:TREE 1523.
INPUT:HUT 2129.
INPUT:NODE 2929.
INPUT:FIELD 1733.
INPUT:FINISH.
DO YOU WANT THE TABLE OF LANDMARKS ?N
DO YOU WANT TO CHANGE TIMESTEP? [BY DEFAULT IT IS TAKEN AS 1 MIN] :Y
```

```
PLEASE TYPE THE TIMESTEP IN FORMAT [<TIME> HR/MIN/SEC ,] :3 MIN.
GIVE TIME OF START OF EXERCISE IN (HR,MIN) 4 FIGURES:1900

YOUR COMMAND:TK 42 MOVE TO 3117 ROUTE 3529,3519.
                                                        TIME:19: 3: 0
ROGER!



YOUR COMMAND:TK 53 MOVE TO 0723 ROUTE 0425.
                                                        TIME:19: 6: 0
ROGER!



TANK  4 2  REPORTING FROM LOCATION  35 24  CANAL  OBSTACLE AHEAD
with MAIN CHARACTEREstics AS FOLLOWS:=

                          DIRECTION   SOUTH WEST  =>>   NORTH EAST
APPROX WIDTH     50  METERS
APPROX DEPTH     25  CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN a "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:GO
TANK  4 2 REPORTING > CANNOT NEGOTIATE THE CANAL - SIDE ARE VERY STEEP


TK  5 3 REPORTING REACHED DESTINATION  723
YOUR COMMAND:TK 13 FIRE 6 SMOKE SHELL LN 0823.
                                                        TIME:19: 9: 0
TK 13 REPORTS GIVING EFFECTIVE FIRE ON LN  823
YOUR COMMAND:TK 22 FIRE 6 SMOKE SHELL LN 3113.
                                                        TIME:19:12: 0
TANK 22 REPORTS GIVING EFFECTIVE FIRE ON LN 3113
YOUR COMMAND:TK 41 MOVE TO 2713 ROUTE 3527,3525,2725.
                                                        TIME:19:15: 0
ROGER!



YOUR COMMAND:TK 51 MOVE TO 0822 ROUTE 0425.
                                                        TIME:19:18: 0
ROGER!
```

```
TK 5 1 REPORTING REACHED DESTINATION  822
YOUR COMMAND:FREEZE.
                                                    TIME:19:21: 0
YOUR COMMAND:TK 13 MOVE TO 1922 ROUTE CROSS.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 12 MOVE TO 1624 ROUTE  TREE.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 23 MOVE TO 2517.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 22 MOVE TO 2717 ROUTE BRIDGE.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 32 MOVE TO 1717 ROUTE 1910,2111.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 33 MOVE TO 1517 ROUTE 0911.
                                                    TIME:19:21: 0
ROGER!

YOUR COMMAND:TK 11 FIRE 8 HE SHELL LN 1123.
                                                    TIME:19:21: 0
YOUR COMMAND:TK 21 FIRE 8 HE SHELL LN 2717.
                                                    TIME:19:21: 0
YOUR COMMAND:CHECK.
                                                    TIME:19:21: 0
THE FOLLOWING CHECKS ARE POSSIBLE
                  LISTING LANDMARKS TYPE "L"
                  LISTING CURRENTLY MOVING TANKS-"T"
                  DRAWINGG THE MAP "M"
                  CHANGING TIMESTEP & TIME "C"
                  SETTING TO DRAW THE MAP-"S"
                  DELETING THE SET MAP DRAWING-"D"
                  RETURN-"R"
CHECK REQUIRED FOR:C
DO YOU WANT TO CHANGE TIME STEP? [BY DEFAULT IT IS TAKEN AS 1 MIN] :N
```

```
GIVE TIME OF START OF EXERCISE IN (HR,MIN) 4 FIGURES:1921
YOUR COMMAND:CONT,
                                                    TIME:19:21: 0



TANK  1 2  REPORTING FROM LOCATION  16 19  RIVER  OBSTACLE AHEAD
witMAIN CHARACTEREstics AS FOLLOWS:-

                          DIRECTION   NORTH WEST  =>>   SOUTH EAST
APPROX WIDTH    100   METERS
APPROX DEPTH     75   CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN A "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:GO


TANK  2 3  REPORTING FROM LOCATION  24 13  RIVER  OBSTACLE AHEAD
with CHARACTEREstics AS FOLLOWS:-

                          DIRECTION   NORTH WEST  =>>   SOUTH EAST
APPROX WIDTH    100  METERS
APPROX DEPTH     75  CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN A "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:GO



YOUR COMMAND:TK 52 MOVE TO 1728 ROUTE 0534,1734.
                                                    TIME:19:43: 0
ROGER!
```

TANK 11 REPORTS GIVING EFFECTIVE FIRE ON LN 1123
TANK 21 REPORTS GIVING EFFECTIVE FIRE ON LN 2717
YOUR COMMAND:TK 53 MOVE TO 0919 ROUTE 0420.

TIME:19:44: 0

ROGER!

TK 1 2 REPORTING REACHED DESTINATION 1624

TK 2 3 REPORTING REACHED DESTINATION 2517

TK 32 REPORTING REACHED DESTINATION 1717

TANK 5 3 REPORTING FROM LOCATION 7 23 RIVER OBSTACLE AHEAD
with MAIN CHARACTEREstics AS FOLLOWS:=

```
                          DIRECTION   NORTH WEST  -->   SOUTH EAST
APPROX WIDTH    100  METERS
APPROX DEPTH     75  CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN A "C" FOR CHANGING COURSE OR A "00" TO CONTINUE:00
TANK 11 REPORTS GIVING EFFECTIVE FIRE ON LN 1123
TANK 21 REPORTS GIVING EFFECTIVE FIRE ON LN 2717
YOUR COMMAND:TK 11 FIRE 8 AP SHELL AT TK 51.
                                                  TIME:19:45: 0




TK  3 3 REPORTING REACHED DESTINATION 1517


YOUR COMMAND:TK 03 FIRE 3 HE SHELL LN 2725.
                                                  TIME:19:46: 0


TK   1 3 REPORTING REACHED DESTINATION 1922

TK   2 2 REPORTING REACHED DESTINATION 2717


YOUR COMMAND:CONT.
                                                  TIME:19:47: 0
```

```
TANK S 3 REPORTING >  FROM LOCATION  620  TANK BOGGED DOWN IN MARSHY GR
TANK 23 REPORTS GIVING EFFECTIVE FIRE ON LN 2725
YOUR COMMAND:TK 23 REPORT VISIBILITY TK 41.
                                                    TIME:19:48: 0

TANKLOCATION        27          20
CANNOT SEE THAT DISTANCE AT NIGHT
IS NOT VISIBLE TO ME
YOUR COMMAND:TK 23 FIRE 4 AP SHELL LN 2722.
                                                    TIME:19:49: 0


YOUR COMMAND:TK 22 FIRE 4 HE SHELL AT TK 41.
                                                    TIME:19:50: 0




YOUR COMMAND:CONT.
                                                    TIME:19:51: 0


TK  4 1 REPORTING REACHED DESTINATION 2713



TK  5 2 REPORTING REACHED DESTINATION 1728
YOUR COMMAND:TK 41 FIRE 5 HE SHELL LN 2617.
                                                    TIME:19:52: 0

YOUR COMMAND:CONT.
                                                    TIME:19:53: 0

TANK 41 REPORTS GIVING EFFECTIVE FIRE ON LN 2617
YOUR COMMAND:TK 23 FIRE 6 HE SHELL AT TK 41.
                                                    TIME:19:54: 0

YOUR COMMAND:CONT.
                                                    TIME:19:55: 0
```

```
TANK 41 REPORTING!UNDER ENEMY HE   FIRE FROM LOC 2517
TANK 23 REPORTS NUETRALISING ENEMY TK-> LOC 2713
YOUR COMMAND:FREEZE.
                                                    TIME:19:56: 0
YOUR COMMAND:TK 13 MOVE TO 1925.
                                                    TIME:19:56: 0
ROGER!

YOUR COMMAND:TK 12 MOVE TO 1927.
                                                    TIME:19:56: 0
ROGER!

YOUR COMMAND:TK 23 MOVE TO 2727.
                                                    TIME:19:56: 0
ROGER!

YOUR COMMAND:TK 22 MOVE TO 3133 !OUTE 2532.
                                                    TIME:19:56: 0
ROGER!

YOUR COMMAND:TK 32 FIRE 10 HE SHELL LN 0923.
                                                    TIME:19:56: 0
YOUR COMMAND:TK 11 MOVE TO 1923.
                                                    TIME:19:56: 0
ROGER!

YOUR COMMAND:CONT.
                                                    TIME:19:56: 0
```

```
TANK  1 2  REPORTING FROM LOCATION  16 24  RIVER  OBSTACLE AHEAD
with MAIN CHARACTEREstics AS FOLLOWS:=

                          DIRECTION   NORTH        =>>    SOUTH
APPROX WIDTH     25   METERS
APPROX DEPTH     25   CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN A "C" FOR CHANGING COURSR OR A "GO" TO CONTINUE:GO
```

```
TANK  1 1  REPORTING FROM LOCATION 15 19 RIVER  OBSTACLE AHEAD
with MAIN CHARACTEREstics AS FOLLOWS:=

                         DIRECTION   NORTH WEST  =>>    SOUTH EAST
APPROX WIDTH   100   METERS
APPROX DEPTH    75   CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:C
TANK ALREADY ON MOVE. DO YOU WANT TO CHANGE STATUS ?Y
FOR CHANGING ONLY DESTINATION TYPE "D"
FOR CHANGING ONLY ROUTE TYPE "R"
FOR COMPLETE CHANGING "C"
FOR RETURN WITHOUT CHANGING "W"
TYPE OF CHANGE REQUIRED ?D
NEW DESTINATION:1525
YOUR COMMAND:CONT.
                                                        TIME:19:57: 0


TK  1 2 REPORTING REACHED DESTINATION 1927


TANK  1 1 REPORTING FROM LOCATION  15 20  RIVER  OBSTACLE AHEAD
with MAIN CHARACTEREstics AS FOLLOWS:=

                         DIRECTION   WEST         =>>    SOUTH EAST
APPROX WIDTH   100   METERS
```

APPROX DEPTH   75   CMS
SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?
TYPE IN A "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:GO
TANK 32 REPORTS GIVING EFFECTIVE FIRE ON LN   923
YOUR COMMAND:TK 13 FIRE 4 AP SHELL AT TK 52.

TIME:19:58: 0

TK 1 3 REPORTING REACHED DESTINATION 1925

TANK 32 REPORTS GIVING EFFECTIVE FIRE ON LN 0923
YOUR COMMAND:TK 52 FIRE 4 HE SHELL LN 1725.

TIME:19:59: 0

TK 1 1 REPORTING REACHED DESTINATION 1525

TANK 52 REPORTING!UNDER ENEMY AP   FIRE FROM LOC 1924
YOUR COMMAND:TK 21 MOVE 2 KM NORTH.

TIME:20: 0: 0

ROGER!

TANK 52 REPORTS GIVING EFFECTIVE FIRE ON LN 1725
YOUR COMMAND:TK 23 FIRE 8 HE SHELL LN 3133.

TIME:20: 1: 0

```
TK   2 3 REPORTING REACHED DESTINATION 2727


YOUR COMMAND:TK 12 FIRE 4 AP SHELL LN 1728.
                                                    TIME:20: 2: 0


TANK 43 REPORTING!UNDER ENEMY HE     FIRE FROM LOC 2625
TANK 23 REPORTS NUETRALISING ENEMY TK-> LOC 3032
TANK 23 REPORTS GIVING EFFECTIVE FIRE ON LN 3133
YOUR COMMAND:CONT.
                                                    TIME:20: 3: 0


TK   2 1 REPORTING REACHED DESTINATION 2515


TANK 43 REPORTING!UNDER ENEMY HE     FIRE FROM LOC 2625
TANK 23 REPORTS GIVING EFFECTIVE FIRE ON LN 3133


TANK 52 REPORTING!UNDER ENEMY AP     FIRE FROM LOC 1927
TANK 12 REPORTS GIVING EFFECTIVE FIRE ON LN 1728
YOUR COMMAND:TK 22 REPORT LOCATION.
                                                    TIME:20: 4: 0
TANK  2 OF TROOP  2   REPORTING

  LOCATION            X         Y
                      25        31


YOUR COMMAND:TK 23 MOVE TO 3133.
                                                    TIME:20: 5: 0
```

ROGER!

TANK 2 3 REPORTING > FROM LOCATION 2828
  TANK HAS RUN INTO ENEMY MINEFEILD!!
YOUR COMMAND:TK 13 FIRE 4 SMOKE SHELL LN 2129.

TIME:20. 6: 0

YOUR COMMAND:TK 12 MOVE TO 2130.

TIME:20: 7: 0

ROGER!

TK 2 2 REPORTING REACHED DESTINATION 3133

TANK 13 REPORTS GIVING EFFECTIVE FIRE ON LN 2129
YOUR COMMAND:TK 13 MOVE TO 2128.

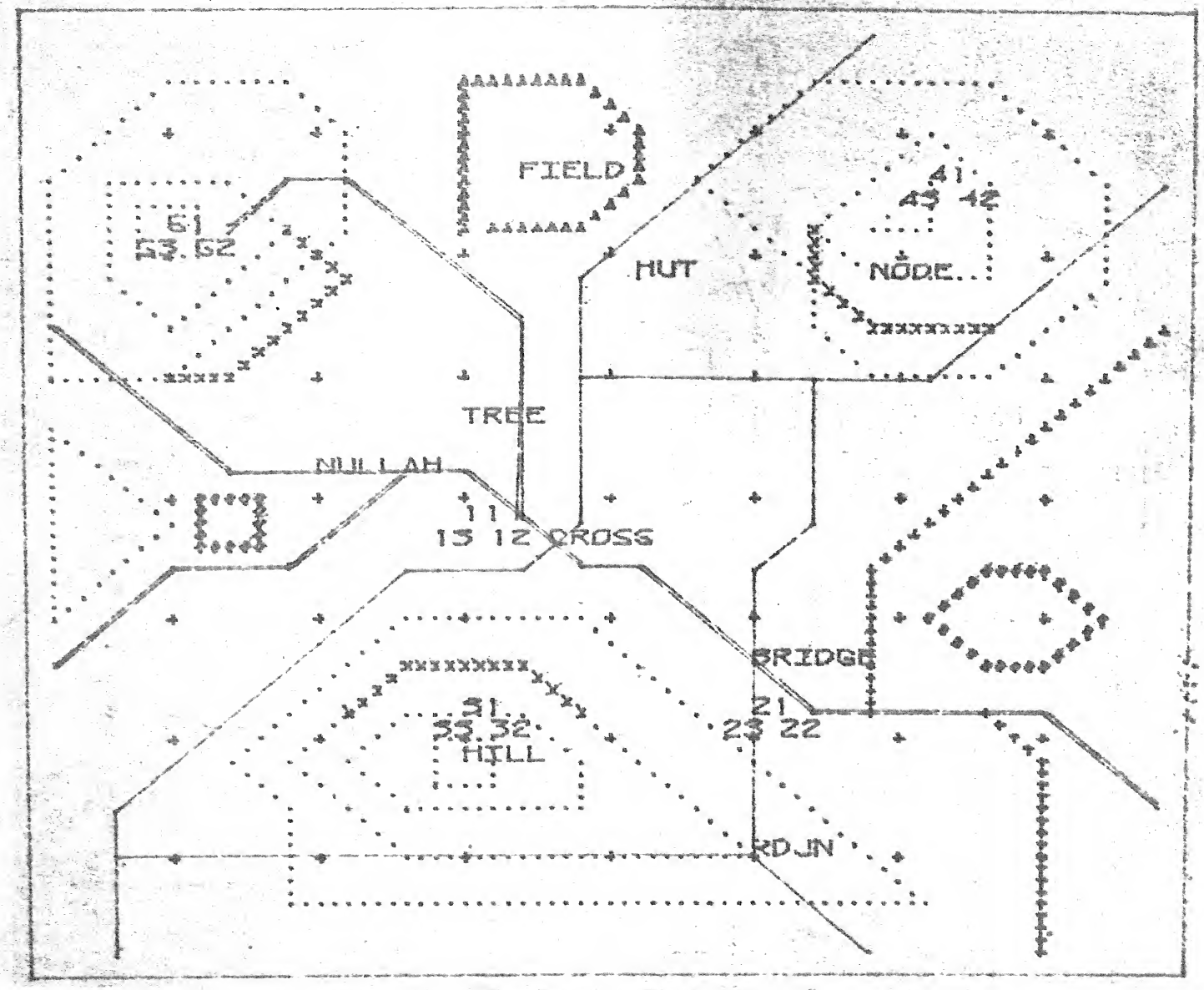
TIME:20: 8: 0

ROGER!

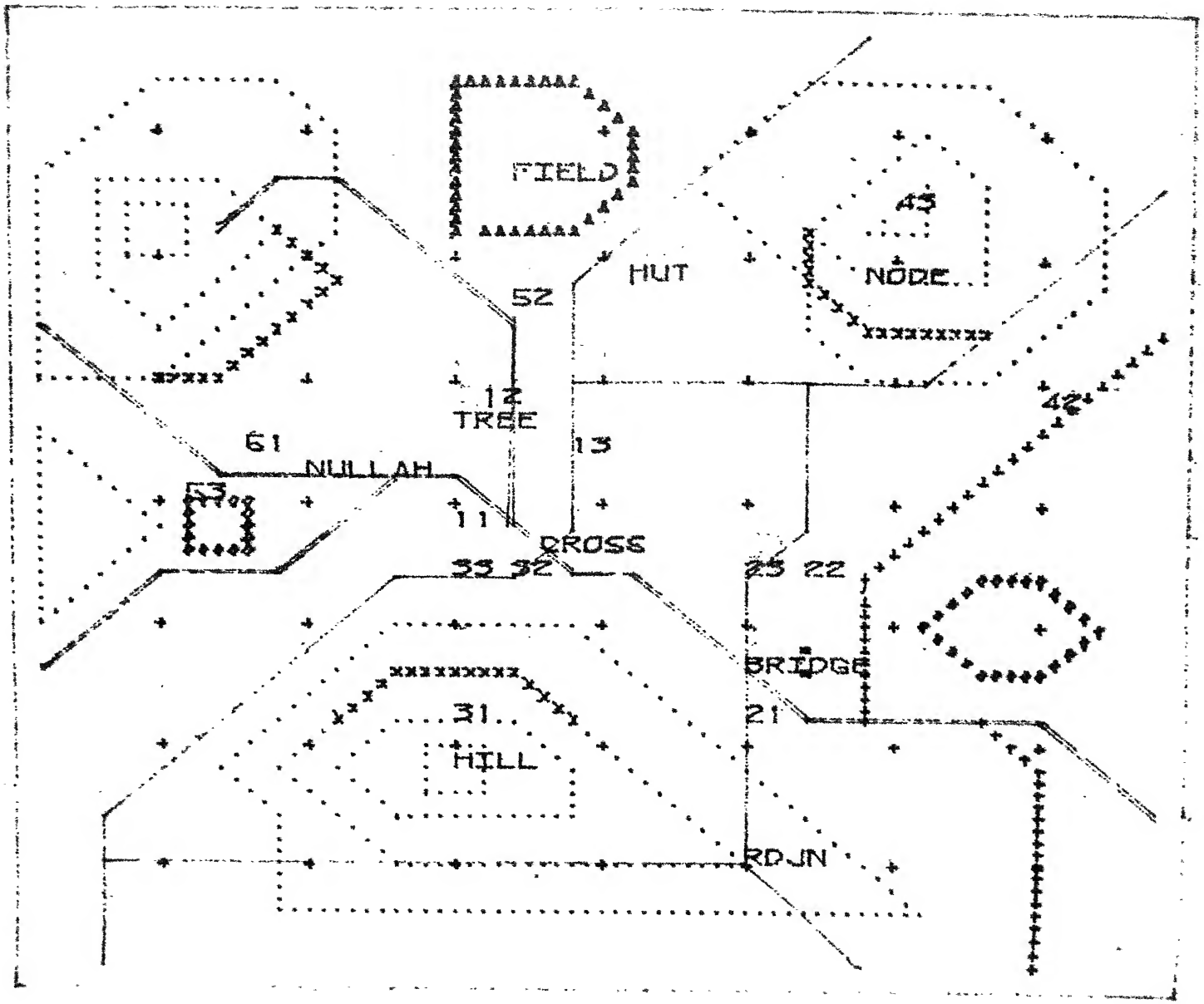
TK 1 3 REPORTING REACHED DESTINATION 2128
YOUR COMMAND:CONT.

TIME:20: 9: 0

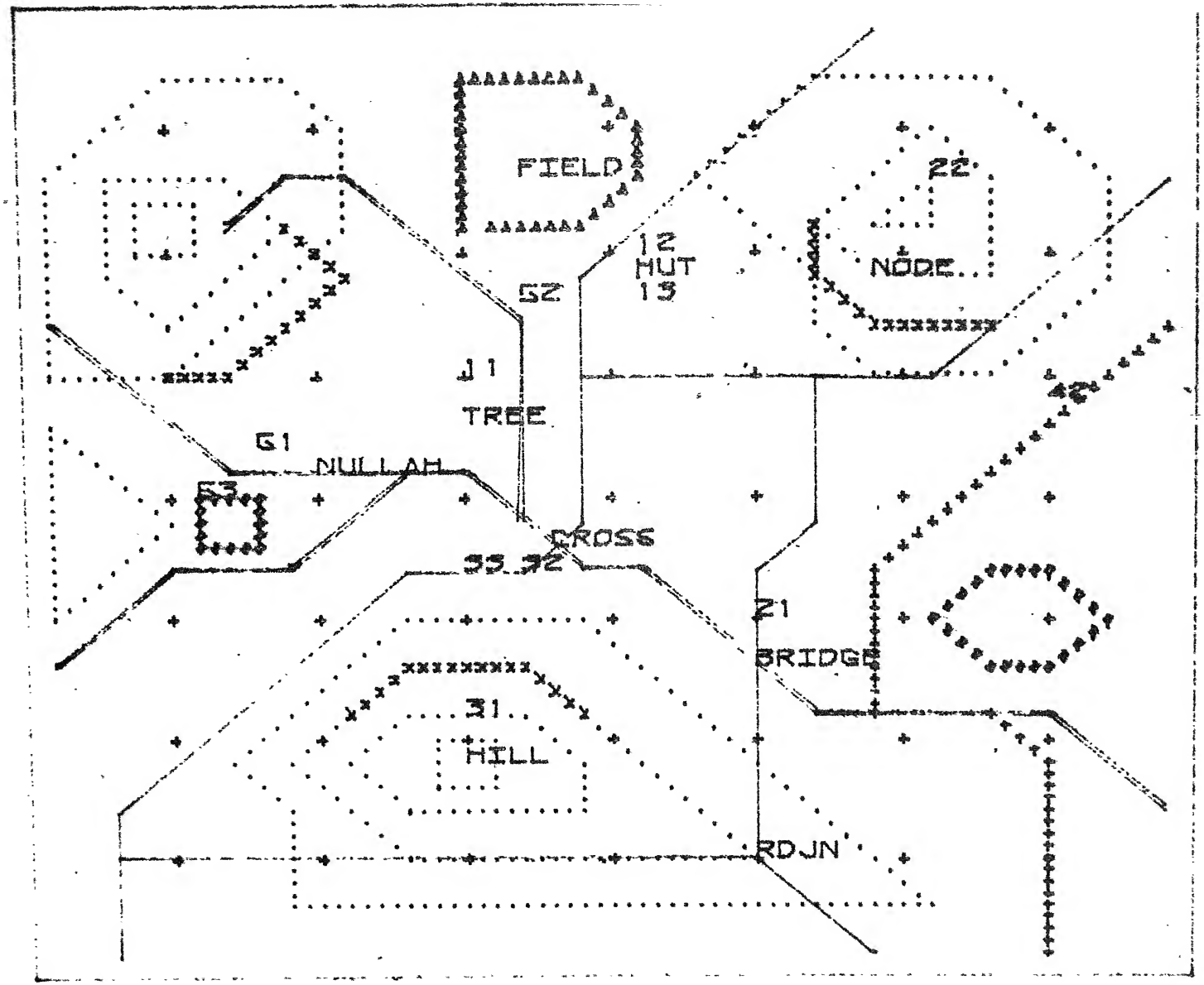
TK 1 2 REPORTING REACHED DESTINATION 2130
YOUR COMMAND:END.


FIELD
61
53.62
HUT
41
43 42
NODE
TREE
NULLAH
11
13 12 CROSS
BRIDGE
31
33.32
HILL
21
23 22
RDJN

MAP - 1


FIELD
HUT
52
43
NODE
12
TREE
61
NULLAH
13
42
53
11
CROSS
33
32
23
22
BRIDGE
31
HILL
21
RDJN

MAP-2


FIELD
12
HUT
13
22
NODE
52
J1
TREE
G1
NULLAH
53
CROSS
33.72
21
BRIDGE
31
HILL
RDJN

MAP - 3

APPENDIX C

```
{*****************************************************************************\
  const
    LASTLIGHT=1050;FIRSTLIGHT=330;
  type
    AMN=record
    TYP1,TYP2,TYP3:integer
    end;
    CORD=record
    X,Y:integer
    end;
    STATREC=record
      BOGGED,MOVING,CANMOV,CANSHOT,FIRING,CUPOLACLOSed,
      NUETRALISEd,SHOCKED:boolean
      end;
    AFVS=record
    TYPNO:1..5;MAPCORD:CORD;ORITASN:0..360;AMMCAP:AMN;
    FUELCAP,ALT,LOADTME:integer;STATUS:STATREC
    end;
    LMPTR=^LMTAB;
    LMTAB=record
      NAME:packed array [1..12]of char;
      GR:integer;
      RLINK,LLINK:LMPTR
    end;
    MPTR=^MOVETAB;
    MOVETAB=record
        ULINK,LLINK:MPTR;
        TANKNO:0..99;
        SOURCE,DEST,LOC:CORD;
        ROUTE:array[1..10]of CORD;
        STATE:boolean;
        OBJECTIVE,CURROBJECTive:integer
      end;
    AFVSCHAR=record
        LENGTH,WIDTH,HEIGHT,
        ARMRFRONT,ARMRSIDE,ARMRREST,
        SPEEDROAD,SPEEDCROSS,SPEEDWATER,
        STEPCLIMB,DITCHCLEAR,
        FUELCAP,FUELRES,MAXGRAD,
        DAYVISIBILity,NIGHTVISIBIlity,
        LPKMCROSS,LPKMROAD:integer
      end;
    AFVSGUNCHar=record
     AMMCAPACITy,(*TOTAL AMMUNATION IT CAN CARRY*)
     MV,(*MUSSLE VELOCITY IN M/SEC*)
     RATEMOVE,(*RATE OF FIRING WHEN TANK ON THE MOVE*)
     RATEHALT,(*RATE OF FIRING WHEN THE TANK IS STATIONERY*)
     RANGEDIRECt,(*RANGE WHEN ENGAGING IN DIRECT FIRE*)
     RANGEINDIRect,(*RANGE WHEN ENGAGING INDIRECT FIRE*)
     ARM500,(*ARMOUR PEIRCING CAPABILITY WHEN DISTANCE OF
          TARGET IS 500 METERS*)
     ARM1000,(*ARMOUR PEIRCING CAPABILITY WHEN DISTANCE OF
          TARGET IS 1000 METEERS*)
     ARM2000,(*ARMOUR PEIRCING CAPABILITY WHEN THE DISTANCE OF
          TARGET IS>2000 METERS*)
     ELEVATION,(*MAXIMUM ELEVATION THAT IS POSSIBLE +/-1*)
```

```
     DEPRESSION:integer(*MAXIMUM DEPRESSION POSSIBLE +/-1*)
    end;
    LETTER=packed array[1..2]of char;
    NAMEWORD=packed array[1..14]of char;
    RANGE=0..8;
    GRIDPTR=^GROUNDCHAR;
    GROUNDCHAR=record
    NEXTPTR:GRIDPTR;
    VALUE1,VALUE2:integer
   end;
    GRIDRECORD=record
    ALTITUDE:integer;
    TYPOFGROUND:0..3;
    GROUNDPTR:GRIDPTR
   end;
    LETTERWORD=packed array[1..12]of char;
    FPTR=^FIRETAB;
    FIRETAB=record
        SOURCETK,DESTTK:packed array[1..2]of integer;
        SOURCELOC,DESTLOC:CORD;
        TURNNO,TIME:integer;
        HITPROB:real;
        RANGE,TYPOFAMN:0..15;
        RLINK,LLINK:FPTR
      end;
    %*****************************************************************\
var
    LINEINPUT:packed array[1..80] of char;(*MAX LENGTH OF INPUT LINE*)
    WORD:packed array [1..12]of char;(*MAX SIZE OF ANY WORD*)
    TANKDAT:array[1..5,1..3]of AFVS;(*TANK IDENTIFICATION, FIRST SUFFIX=TANKNUMBER;
                    SECOND SUFFIX=TROOP NUMBER*)
    K,INTVAL,IK,JK,I,J,L,M:integer;
    NUMWORD:packed array[1..4]of 0..9;(*FOUR FIGURE GRID REFERENCE*)
    F,ISNUMBER,T:boolean;
    MR,MP,MQ,MLAST,MOVTABROOT:MPTR;(*POINTERS TO MOVE TABLE*)
    DEG:integer;
    ISLANDMARK:boolean;
    CH:char;
    LP,LQ,LR,LMROOT:LMPTR;(*POINTERS TO LANDMARKS TABLE*)
    TANKNUMBER:integer;
    TESTNUMBER:boolean;
    A,B:integer;
    DIST:integer;
    ANGLE:real;
    ISINMOVTAB,MAPREQUIRED,CROSSED:boolean;
    TIMESTEP,TIMESEC,TIMEMIN,TIMEHR:integer;(*USED FOR TIME STEP CALCULATION*)
    FREEZE:boolean;
    TANKCHAR:array[1..3]of AFVSCHAR;
    GUNCHAR:array[1..3]of AFVSGUNCHAR;
    GRX,GRY:LETTER;
    GRIDCHAR:array[0..41,0..41]of GRIDRECORD;TEMPPTR,PTR:GRIDPTR;
    MAPDAT,MAP,FIREOT:file of char;
    REACHEDESTination:boolean;
    TYPOFTANK:1..4;
    ERRORPOINT:integer;
    FROOT,FLAST,FP,FQ,FR:FPTR;
```

```
JOIN:boolean;
AIMUPKSTEP:integer;
FIRERRPORT:array[1..5,1..3] of boolean;
HIT:array[1..5,1..3] of boolean;
NAME:ANEWORD;PAPER:packed array[1..4]of char;
%***********************************************************************\
procedure ERROR( J:integer);(*PRINTS THE TYPE OF ERROR WITH MESSAGES*)
(* used for pointing out command errors during execution of programme *)
%***********************************************************************\
var
    J:integer;
begin
    WRITELN(TTY,LINEINPUT);for J:=1 to ERRORPOINT do WRITE(TTY,' ');
    WRITELN(TTY,'^');WRITELN(TTY);
    WRITE(TTY,'ERROR IN INPUT COMMAND-FOUND >  ',WORD,'  <WHEN EXPECTING  ');
    WRITELN(TTY);
    case J of
    1:WRITELN(TTY,'STARTING SYMBOL TK');
    2:WRITELN(TTY,'TK  AT START OF A COMMAND');
    3:WRITELN(TTY,'STATUS,LOCATION,BALANCE,ENEMY,VISIBILITY');
    4:WRITELN(TTY,'<TANKNAME MUST END WITH ,/BLANK');
    5:WRITELN(TTY,'LESS THAN FOUR INTEGERS IN GRID REFERENCE');
    6:WRITELN(TTY,'INTEGERS IN GRID REFERENCE');
    7:WRITELN(TTY,'LESS THAN 80 CHAR IN ACOMMAND');
    8:WRITELN(TTY,'  AMMUNATION OR FUEL AFTER BALANCE');
    9:WRITELN(TTY,' <,> OR <TK> AFTER STATUS');
    10:WRITELN(TTY,' <LM> OR <TK> AFTER VISIBILITY');
    11:WRITELN(TTY,'UNIQUE LANDMARK OR GRIDREFERENCE');
    12:WRITELN(TTY,'LESS THAN 20 LANDMARKS OR INTEGERS IN GRID REF');
    14,18,21:WRITELN(TTY,'LANDMARK/GRIDREFERENCE');
    15:WRITELN(TTY,'"."/"OF"/<DIST> AFTER MOVE');
    16,13:WRITELN(TTY,'<360 DEGREE IN DIRECTION');
    17:WRITELN(TTY,'MTR,MTRS,METER,KM,KMS');
    20:WRITELN(TTY,'"ROUTE"/"." AFTER OBJ');
    22:WRITELN(TTY,'NO PREVIOUS COMMAND');
    23:WRITELN(TTY,'SMOKE/HE/AP ie type of ammunation');
    24:WRITELN(TTY,'TANKNAME/TANKID');
    25:WRITELN(TTY,'TK/TANK after AT');
    26:WRITELN(TTY,'AT/GR after SHELL(S)');
    27:WRITELN(TTY,'SHELL(S) after typ of ammunation')
    end;
    BREAK
end;
%***********************************************************************\
procedure CHECKPTS;(*GENERAL GUIDELINES TO RUN THE PROGRAMME*)
%***********************************************************************\
begin WRITE(TTY,'THIS IS A PROGRAMME EXERCISE FOR TANK BATTLE AT SQUADRON LEVEL.');
    WRITE(TTY,' YOU ARE REQUIRED TO TYPE IN COMMAND IN THE FORMAT SPECIFIED ');
    WRITE(TTY,'AND IT WILL BE EXECUTED. ');
    WRITELN(TTY,'THE FOLLOWING POINTS NEED PARTICULAR ATTENTION:-');
    WRITELN(TTY,'1':10,' THE COMMAND MUST BE TYPED STARTING
WITH <TK> OR OTHER COMMAND WORDS');
    WRITELN(TTY,'2':10,' THE COMMAND MUST END WITH <.>');
    WRITELN(TTY,'3':10,' IN CASE OF HELP TYPE <HELPCOMMD> OR <HELP>');
    WRITELN(TTY,'3':10,' TO TERMINATE THE PROGRAMME TYPE <END.>');
    WRITELN(TTY,'4':10,' AFTER YOU HAVE WRITTEN YOUR
```

```
01690       COMMAND GIVE CARRAIGE RETURN');
01700             WRITELN(TTY,'S':10,' IF YOU WANT THESE COMMANDS TO BE
01710       TYPED THEN TYPE COMMAND <CARRIES.>');
01720             WRITELN(TTY,'READY TO START NOW');
01730             BREAK
01740         end;
01750             %*****************************************************************\
01760         procedure MSG(*:integer);(*PRINTS MESSAGES*)
01770         (* this is primarily used to print out messages originating from the programme *)
01780         %*****************************************************************\
01790         begin
01800             case * of
01810             1:WRITELN(TTY,'THIS TANK ALREADY DID MOVE');
01820             2:WRITELN(TTY,'EXPECTING A D/R/C/W HENCE "W" ASSUMED');
01830             3:WRITELN(TTY,'NOT 3 CHARS MORE THAN 12 BEING IGNORED');
01840             4:WRITELN(TTY,'TANK BOGGED CANNOT MOVE');
01850             5:WRITELN(TTY,'CANNOT DO THE MOVE');
01860             6:WRITELN(TTY,'TANK GUN FIRING ON THE TARGET');
01870             7:WRITELN(TTY,'TANK GUN OUT OF ACTION - CANNOT FIRE');
01880             8:WRITE(TTY,'TANK CANNOT MOVE DUE TO ');
01890             9:WRITELN(TTY,'HIT BY ENEMY FIRE');
01900             10:WRITELN(TTY,'MECHANICAL FAILURE');
01910             11:WRITELN(TTY,'DIRECTION WRONG-GOING OUT OF MAP AREA');
01920             12:WRITE(TTY,'TANK',IK:2,JK:2,' REPORTING > ');
01930             13:WRITELN(TTY,'TANK HAS RAN OUT OF FUEL');
01940             14:WRITELN(TTY,'TANK RUNNING ON RESERVE');
01950             15:WRITELN(TTY,'WATER TOO DEEP - TANK IS BOGGED');
01960             16:WRITELN(TTY,'CANNOT NEGOTIATE THE CANAL - SIDES ARE VERY STEEP');
01970             17:WRITELN(TTY,'SHOULD HAVE GIVEN DIRECTION ie BACK/LEFT etc');
01980             18:WRITELN(TTY,'SORRY! NOT ABLE TO SHOOT-IK',IK:1,JK:1);
01990             19:WRITELN(TTY,'CANNOT SEE AT THAT DISTANCE AT NIGHT');
02000             20:WRITELN(TTY,'CANNOT SEE THAT DISTANCE IN DAY');
02010             21:WRITELN(TTY,'LOOKS OF COVER AT THAT LOCATION')
02020             end;
02030             BREAK
02040         end;
02050             %*****************************************************************\
02060         procedure HELPREPORT;(*PRINTS COMMANDS AVAILABLE IN 'REPORT PROCEDURE'*)
02070         (* lists all possible commands available in REPORT command - used in the case of HEL
02080         %*****************************************************************\
02090         begin
02100             WRITELN(TTY,' the following commands are possible in report');
02110             WRITELN(TTY,'command with appropriate options');
02120             WRITELN(TTY,'  ':5,'(tk <tankname>) report location.');
02130             WRITELN(TTY,'  ':27,'status tk <tankname>/s  loc <grid ref>/$ .');
02140             WRITELN(TTY,'  ':27,'balance ammunation/fuel he/smoke/ap/all  .');
02150             WRITELN(TTY,'  ':27,'enemy .');
02160             WRITELN(TTY,'  ':27,'visibility tk <tankname>/loc <grid ref> .');
02170             WRITELN(TTY,'  ':10,' note:<tankname>::=tank no followed by troop no:');
02180             WRITELN(TTY,'  <grid REF>::=six figure grid ref') ;
02190             BREAK
02200         end;
02210             %*****************************************************************\
02220         procedure COMMANDHELP;
02230         %*****************************************************************\
02240         begin
```

```
02250           WRITELN(TTY,'THE FOLLOWING COMMANDS ARE POSSIBLE AS "COMMANDS"');
02260           WRITELN(TTY,'  <no> <firepart>/<movepart>/<reportpart>.');
02270           WRITELN(TTY,'HELP/HELPCOMND.');
02280           WRITELN(TTY,'LANDMARK <gives the listing of defined landmarks>');
02290           WRITELN(TTY,'TANKTAB <gives the listing of tanks');
02300           WRITELN(TTY,' of the move with details>' );
02310           WRITELN(TTY,'FREEZE <freezes all ACTION part for the duration>');
02320           WRITELN(TTY,'START/CONTINUE <continues the ACTION part after freeze>');
02330           WRITELN(TTY,'END. <stops the programme completely>');
02340       end;
02350           {*************************************************************\
02360       procedure HELPMOVE;(*LISTS COMMANDS AVAILABLE IN 'MOVE' PROCEDURE*)
02370       (* lists all possible commands available in MOVE command - used in the case of HELP
02380       {*************************************************************\
02390       begin
02400           WRITE(TTY,'THE FOLLOWING COMMANDS ARE POSSIBLE IN "MOVE"');
02410           WRITELN(TTY,'  ALONGWITH APPROPRIATE OPTIONS:');
02420           WRITELN(TTY,'(AS <TANKNAME>)MOVE.');
02430           WRITELN(TTY,'  :10,'TO LANDMARK.');
02440           WRITELN(TTY,'  :20,'ROUTE <LANDMARKS>.');
02450           WRITELN(TTY,'  :10,'<DISTANCE> MTR/METER <DIRECTION>.');
02460           WRITELN(TTY,'  :50,'OF <LANDMARK>.');
02470           WRITELN(TTY,'  :50,'ROUTE <LANDMARKS>.');
02480           WRITELN(TTY,'DIRECTION:cardinal-direct::=north/n,north-east/ne,etc.');
02490           WRITELN(TTY,'  :10,'direction::=clock-direction/cardinal-direction/degree');
02500           WRITELN(TTY,'  :10,'clock-direct::=12 O"clock,1 O"clock etc.');
02510           WRITELN(TTY,'  :10,'landmark::=<defined landmark name>');
02520           WRITELN(TTY,'  :10,'landmarks::=<landmark>,<landmark>.');
02530       end;
02540           {*************************************************************\
02550       procedure HELPFIRE;(* PRINTS COMMANDS THAT ARE AVAILABLE IN FIRE COMMAND*)
02560       {*************************************************************\
02570       begin
02580           WRITELN(TTY,'the following commands are possible in FIRE commands');
02590           WRITELN(TTY,'  :50,'OK <no> FIRE/SHOOT <no> HE/AP/SMOKE SHELL(S) AT OK <no>.');
02600           WRITELN(TTY,'  :50,'<direction>.');
02610           WRITELN(TTY,'  :50,'OK <gridref>.');
02620           WRITELN(TTY,'  :53,'OK <gridref>.');
02630           WRITELN(TTY,'  :10,'STOP FIRING');
02640           WRITELN(TTY,'NOTE->direction::=north/front/right/left etc');
02650       end;
02660           {*************************************************************\
02670       procedure READCH;(*READS A CHARACTER FROM THE INPUT COMMAND*)
02680       (* reads the first char from the input command and
02690     stores in CH. used for yes/no type of dialogue *)
02700       {*************************************************************\
02710       var
02720           ALP:char;
02730       begin
02740           repeat READ(TTY,CH)
02750           until CH#' ';
02760           while not EOLN(TTY) do READ(TTY,ALP);
02770       end;
02780           {*************************************************************\
02790       procedure TYPEHELP;(*DECIDES TYPE OF HELP REQUIRED*)
02800       (* in case of HELP implies the type of HELP required *)
```

```
02810         %*************************************************************\
02820         begin
02830             READCH;
02840             case CH of
02850             'R':HELPREP(RR);
02860             'M':HELPMOVE;
02870             'F':HELPFIRE;'C':GOTO HELP;
02880             'E':;
02890             others:
02900               begin WRITELN(TTY,'your>',CH,'<not recognised please retype:');
02910                     BREAK;READCH
02920               end
02930             end
02940         end;
02950             %*************************************************************\
02960         procedure HELPCOMMAND;(*PRINTS 'HELP' IN COMMAND PROCEDURE*)
02970         %*************************************************************\
02980         begin
02990             WRITELN(TTY,'yes! this is helper for your commands,
03000         please type char for the help required:-');
03010             WRITELN(TTY,' E ',' for return to execution :E ');
03020             WRITELN(TTY,' R ',' for report commands :R ');
03030             WRITELN(TTY,' M ',' for move commands :M ');
03040             WRITELN(TTY,' F ',' for fire commands :F ');
03050             WRITELN(TTY,' C ',' for COMMAND commands :C ');
03060             WRITE(TTY,' ',' help required for:');BREAK;TYPEHELP
03070         end;
03080             %*************************************************************\
03090         procedure READLINE;(*READS IN ONE LINE INPUT AND STORES IN 80 LINE BUFFER*)
03100         (* reads upto 80 chars from the tty to process the command.
03110            it is terminated by "."or EOLN or >80 char *)
03120         %*************************************************************\
03130         var
03140             IL:integer;ALP:char;
03150         begin  for IL:=1 to 80 do LINEINPUT[IL]:=' ';
03160             ERRORPOINT:=1;
03170             IL:=1;
03180             repeat READ(TTY,ALP);LINEINPUT[IL]:=ALP;
03190             IL:=IL+1
03200             until ((ALP='.')or(IL>80)or(EOLN(TTY)));
03210             if ALP='.' then
03220               begin WRITE(TTY,'MISSING POINT AT END PLEASE RETYPE:');
03230             BREAK;READLINE
03240               end
03250             else
03260             I:=1
03270         end;
03280             %*************************************************************\
03290         procedure READWORD;(*READS ONE WORD FROM THE INPUT COMMAND*)
03300         (* takes the next word from the input command for processing. input to
03310            this from the buffer LINEINPUT *)
03320         (* delimiters between words are "b","," and for termination
03330            in addition a "." is included. *)
03340         (* it also stores the last word location in ERRORPOINT for error messages *)
03350         %*************************************************************\
03360         var
```

```
    J:integer;
begin for J:=1 to 12 do WORD[J]:=' ';
    J:=1;while((LINEINPUT[I]=' ')or(LINEINPUT[I]=',')) do I:=I+1;
    ERRORPOINT:=I;
    if I<80 then
      begin
    repeat
    if J<13 then WORD[J]:=LINEINPUT[I];
    J:=J+1;I:=I+1
    until
    ((LINEINPUT[I]=' ')or(LINEINPUT[I]='(')or(LINEINPUT[I]=')')
    or(LINEINPUT[I]=',')or(WORD[J-1]='(')
    or(WORD[J-1]=')')or(LINEINPUT[I]=','));
    if J>12 then
    ISS(3)
      end
    else ERROR(7)
end;
     %*************************************************************************\
procedure READINPUT;(*READS DIRECTLY INPUT UPTO 12 CHARACTERS*)
(* reads directly 12 char from tty and stores them in WORD.
used to interactive communication    *)
%*************************************************************************\
var
    INCH:char;I:integer;
begin
    for I:=1 to 12 do WORD[I]:=' ';
    INCH:=' ';I:=1;
    while INCH=' ' do READ(TTY,INCH);
    while ((not EOLN(TTY))and(I<13)and(INCH#'.')) do
      begin
    WORD[I]:=INCH;I:=I+1;READ(TTY,INCH);
      end;
    if I>12 then ISS(3)
    else
    if EOLN(TTY) then WORD[I]:=INCH
end;
     %*************************************************************************\
procedure NUMBER;(*CONVERTS CHAR NUMBER TO INTEGER*)
(* this procedure checks if the current word in the buffer WORD is an integer.
 if testing is set by TESTNUMBER then    sets this *)
(* to TRUE if it is a NUMBER and converts it to integer return in INTVAL *)
%*************************************************************************\
var
    M,N:integer;CH:char;
begin for M:=1 to 4 do NUMWORD[M]:=0;
    M:=1;INTVAL:=0;ISNUMBER:=true;
    loop
    CH:=WORD[M];
    exit if ((CH=' ')or(M>4)or(ORD(CH)>ORD('9'))or(ORD(CH)<ORD('0')));
    N:=ORD(CH)-ORD('0');NUMWORD[M]:=N;
    M:=M+1;INTVAL:=INTVAL*10+N
    end;
    if ((CH#' ')and(M>4)) then ERROR(5);
    if ( CH#' ') then
      begin
```

```
        ISNUMBER:=false;
        if TESTNUMBER then
          begin
        ERROR(6);WRITE(TTY,'WOULD YOU LIKE TO SUBSTITUTE THE NUMBER ?:');
        BREAK;READCH;
        if CH='Y' then
          begin
            WRITE(TTY,'YOUR NEW NUMBER:');BREAK;READWORD;NUMBER?
          end;
          end
          end;
       TESTNUMBER:=false
end;
   %***********************************************************************\
procedure SKIP(P:integer);(*SKIPS 'P' LINES WHILE OUTPUTTING*)
(* used to skip lines on tty for clearity of presentation *)
%**********************************************************************\
var
    J:integer;
begin for J:=1 to P do WRITELN(TTY)
end;
   %***********************************************************************\
function YES:boolean;(*CHECKS ANS*)
%**********************************************************************\
begin
    READCH;
    if CH='Y' then YES:=true
    else YES:=false
end;
   %***********************************************************************\
function REDUCE(X:integer):real;(*reduces the scale from integer to real for graphic
%**********************************************************************\
begin
    REDUCE:=X/40
end;
   %***********************************************************************\
procedure CONVERT(INT:integer;var A,B:integer);
(*SPLITS THE GRIDREF TO NORTHINGS AND EASTINGS*)
(* INT is the input value and A,B give the northings and eastings of the location *
%**********************************************************************\
begin
    B:=INT mod 100;
    A:=INT div 100
end;
   %***********************************************************************\
procedure CHECKGANDMARK(var GM:integer);
    forward;
   %***********************************************************************\
procedure CALCULATEDist(X1,Y1,X2,Y2:integer;var DISTANCE:integer);
(* calculates the distance between grdref x1,y1 and x2,y2 and outputs it in DISTANC
%**********************************************************************\
var
    TEMP:real;
begin
    TEMP:=((X1-X2)*(X1-X2) + ((Y1-Y2)*(Y1-Y2)));
    DISTANCE:=TRUNC(SQRT(TEMP)+0.5)
```

```
end;
   (*************************************************************************\
procedure PRTLMTAB(P:LMPTR);(*PRINTS TABLE OF LANDMARK ENTRIES*)
(* it does the inorder printing of LANDMARKS defined by the user and stored in [illegible]
 and accesed through LMPTR *)
(**************************************************************************\
begin
   if P=LMROOT then WRITELN(TTY,'LANDMARK NAME':25,'GRID REF':20);
   if (P^.LLINK#nil) then PRILMTAB(P^.LLINK);
   WRITELN(TTY);
   if P#LMROOT then
   WRITELN (TTY,P^.NAME:25,P^.GR:20);
   if (P^.RLINK#nil) then PRILMTAB(P^.RLINK)
end;
   (*************************************************************************\
procedure ENTERLM;(*UPDATES/ADDS TO LANDMARK TABLE*)
(**************************************************************************\
var
   R:boolean;I:integer;
begin
   WRITE(TTY,'DO YOU WANT TO DEFINE LANDMARKS ?');BREAK;
   READCH;I:=0;
   if CH='Y' then
     begin
   WRITELN(TTY,'TYPE IN LANDMARKS IN THE FOLLOWING FORMAT [<LM>,<GR>.]');
   WRITELN(TTY,'WHEN YOU HAVE FINISHED then type [FINISH.]');
   loop
   I:=I+1;WRITE(TTY,'INPUT:');BREAK;READLINE;READWORD;
   if WORD#'FINISH     ' then
     begin NEW(LQ);LQ^.NAME:=WORD;READWORD;NUMBER
     end;
   R:=true;
   exit if ((WORD='FINISH     ')or(I>20)or(ISNUMBER=false));
   LQ^.GR:=INTVAL;LQ^.LLINK:=nil;LQ^.RLINK:=nil;
   if LP=LMROOT then
     begin
        LP:=LQ;LMROOT^.RLINK:=LP
     end
   else
     begin
        LP:=LMROOT^.RLINK;
        while ((LP#nil)and(R=true)and(LP^.GR#LQ^.GR)) do
     begin
        LR:=LP;
        if LQ^.NAME=LP^.NAME then R:=false
        else
          begin
        if LQ^.NAME>LP^.NAME then LP:=LP^.RLINK
        else LP:=LP^.LLINK
          end
     end;
   .    if LP=nil then
     begin
        if LQ^.NAME>GR^.NAME then GR^.RLINK:=LQ
        else GR^.LLINK:=LQ
     end
```

```
            else ERROR(11)
        end
    end;
    if (WORD#'FINISH      ') then ERROR(12)
        end;
    WRITE(TTY,'DO YOU WANT THE TABLE OF LANDMARKS ?');
    BREAK;
    READCH;
    if CH='Y' then
    PRINTTAB(LANDTOP);
end;
    %*************************************************************************\
procedure READTIMESTEP;(*READS IN TIMESTEP TO BE USED*)
%*************************************************************************\
var
    A,B:integer;
begin
    FREEZE:=false;
    WRITE(TTY,'DO YOU WANT TO CHANGE TIMESTEP? (BY  DEFAULT IT IS TAKEN AS 1 MIN) :');
    BREAK;READCH;
    if CH='Y' then
        begin
    WRITE(TTY,'PLEASE TYPE TIMESTEP IN FORMAT [<TIME> HR/MIN/SEC .] :');BREAK;
    READLINE;READWORD;NUMBER;
    if ISNUMBER then
        begin
    READWORD;
    if WORD='MIN        ' then TIMESTEP:=INTVAL*60
    else
    if WORD='HR         ' then TIMESTEP:=INTVAL*3600
    else
    if WORD='SEC        ' then TIMESTEP:=INTVAL
        end
        end
    else TIMESTEP:=60;
    WRITE(TTY,'GIVE TIME OF START OF EXERCISE IN (HR,MIN) 4 FIGURES:');BREAK;
    READINPUT;NUMBER;CONVERT(INTVAL,A,B);
    TIMESEC:=A*3600+B*60;TIMEHR:=0;
    MINTIMESTEP:=TIMESTEP div 60;
end;
    %*************************************************************************\
procedure INITGRID;(*initialises the grid poits to plain ground and rest to 0 *)
%*************************************************************************\
var
    I,J:integer;
begin
    for I:=1 to 40 do
    for J:=1 to 40 do
    with GRIDCHAR[I,J] do
        begin
            ALTITUDE:=0;
            TYPOFGROUND:=8;
            GROUNDPTR:=nil
        end;
end;
    %*************************************************************************\
```

```
05610       procedure CALCNEXTGRIDPOINT(var X,Y:integer;DIRECTION:RANGE);
05620       (* given the DIRECTION and the present location in X,Y
05630       it gives the next gridpoint in X,Y *)
05640       {*****************************************************************\
05650       begin
05660           case DIRECTION of
05670           0:Y:=Y+1;
05680           1:
05690             begin X:=X+1;
05700                 Y:=Y+1;
05710             end;
05720           2:
05730           X:=X+1;
05740           3:
05750             begin
05760                 X:=X+1;
05770                 Y:=Y-1;
05780             end;
05790           4:
05800           Y:=Y-1;
05810           5:
05820             begin X:=X-1;
05830                 Y:=Y-1
05840             end;
05850           6:
05860           X:=X-1;
05870           7:
05880             begin
05890                 X:=X-1;
05900                 Y:=Y+1
05910             end
05920           end;
05930       end;
05940       {*****************************************************************\
05950       procedure INITIALISE;
05960       (* initialises the details for the complete programme *)
05970       {*****************************************************************\
05980       begin
05990           RESET(MAPDAT);
06000           REWRITE(FIREDI);
06010           RTTOPEV(1);
06020           CHECKPTS;
06030           NEW(LMROOT);
06040           LMROOT^.RLINK:=nil;LMROOT^.LLINK:=nil;LP:=LMROOT;LMROOT^.TAG:=
06050           TESTNUMBER:=true;
06060           NEW(MOVTABROOT);MOVTABROOT^.LLINK:=nil;P:=MOVTABROOT;TAG:=P;
06070           MP^.LLINK:=nil;ISGAMMARK:=false;
06080           NEW(FROOT);FLAST:=FROOT;FROOT^.LLINK:=nil;FROOT^.RLINK:=nil;
06090           MAPREQUIRED:=false;
06100           with TANKCHAR[1] do
06110             begin
06120           LENGTH:=900;(*CM*)
06130           WIDTH:=327;(*CM*)
06140           HEIGHT:=235;(*CM*)
06150           ARMFRONT:=200;(*MM*)
06160           ARMSIDE:=160;(*MM*)
```

```
        ARTRREST:=60;(*[illegible]*)
        SPEEDROAD:=50;(*KMPH*)
        SPEEDCROSS:=30;(*KMPH*)
        SPEED AIFR:=[illegible];(*KMPH*)
        STEPCLIMB:=[illegible];(* [illegible] .1 METER *)
        DITCHCLEAR:=27;(* [illegible] .1 METER *)
        FUELCAP:=520;(*LITRES*)
        FUELRES:=50;(*LITRES*)
        MAXGRAD:=32;(*DEGREES*)
        LPKMCROSS:=3;(*LITRES PER KM*)
        DAYVISIBILITY:=[illegible];
        NIGHTVISIBILITY:=5;
        LPKMROAD:=2;(*LITRES PER KM ON ROAD *)
          end;
        with GUNCHAR[1] do
          begin
        AMMCAPACITY:=[illegible];(*RNDS*)
        MV:=900;(* [illegible] *)
        RATEMOVE:=4;(* ROUNDS PER MINUTE*)
        RATEHALT:=7;(* ROUNDS PER MINUTE*)
        RANGEDIRECT:=7[illegible];(*METERS*)
        RANGEINDIRECT:=14600;(*METERS*)
        AR[illegible]500:=200;(*[illegible]*)
        AR1000:=185;(*[illegible]*)
        AR2000:=155;(*[illegible]*)
        ROTATION:=1[illegible];(*DEGREES*)
        DEPRESSION:=[illegible];(*DEGREES*)
          end
    end;
        %*****************************************************************************\
    procedure INITTANK;
    (* initialises the various location and other details regarding the type of tank
     used in the programme. the details are taken interactively from the user*)
    %*****************************************************************************\
    var
        I,ADD,J,X,Y,AMN1,AMN2,AMN3:integer;
    begin
        repeat
        WRITELN(TTY,'PLEASE GIVE THE TYPE OF TANK USED - ');
        WRITELN(TTY,'if it is VIJAYANTA then type "V"');
        WRITELN(TTY,'if it is T-55 then type "T"');
        WRITE(TTY,'type of tank :');BREAK;
        READCH;
        case CH of
        'V':
          begin
              TYPOFTANK:=1;
              AMN1:=10;AMN2:=20;AMN3:=13
          end;
        'T':
          begin
              TYPOFTANK:=1;
              AMN1:=10;AMN2:=20;AMN3:=13
          end;
        others:WRITELN(TTY,'YOUR ',CH,' NOT RECOGNISED PLEASE RETYPE -')
        end;
```

```
until ((CH='Y')or(CH='N'));
WRITELN(TTY,'PLEASE GIVE THE LOCATION OF VARIOUS TROOPS OF YOUR SQUADRON');
WRITELN(TTY,'YOU ARE REQUIRED TO GIVE ONLY FOUR FIGURE GRID REFERENCES');
for J:=1 to 5 do
  begin
  WRITE(TTY,'TROOP NO',J:3,':');BREAK;
  READINPUT;GRIDCONVERT(INTVAL,X,Y);
  ADD:=1;
  for I:=1 to 3 do
  with TKDATA[J,I] do
    begin
        ORGTASK:=TYPNO:=TYPOFTANK;
        FUELCAP:=TANKCHAR[TYPOFTANK].FUELRES + TANKCHAR[TYPOFTANK].
        FUELCAP;
        LOOPNO:=1;
        if I=1 then
    begin MAPCORD.X:=X;MAPCORD.Y:=Y;
    end
        else
    begin
      MAPCORD.X:=X+ADD;MAPCORD.Y:=Y-1;ADD:=ADD*(-1);
    end;
        AMMCAP.TYP1:=AMM1;
        AMMCAP.TYP2:=AMM2;AMMCAP.TYP3:=AMM3;ALT:=GRIDCHAR[X,Y].ALTITUDE;

        with STATUS do
    begin
        BOGGED:=false;MOVING:=false;
        FIRING:=false;CUPOLACLOSed:=true;
        GENERALISEd:=false;CANMOV:=true;
        CANSHOOT:=true;SHOCKED:=false
    end
    end
    end
end;
   %**********************************************************************\
procedure CONVERTGRID(var GRX,GRY:LETTER;X,Y:integer);
(* converts the integer cordinates "x","y" to grid ref x,y
   and outputs  GRX,GRY as chars *)
(* these are used for storage and in printing out messages *)
%*********************************************************************\
var
   I,J,X1,Y1:integer;
begin
   X1:=X;Y1:=Y;
   for I:=2 downto 1 do
     begin
   J:=(X1)mod(10);X1:=(X1)div(10);GRX[I]:=CHR(J+ORD('0'));
   J:=(Y1)mod(10);Y1:=(Y1)div(10);GRY[I]:=CHR(J+ORD('0'));
     end
end;
   %**********************************************************************\
procedure UPDATETIME;(*UPDATES THE TIME ELAPSED IN HR,MIN,SEC*)
%*********************************************************************\
begin
   if not FREEZE
```

```
        then
          begin
        TIMESEC:=TIMESEC+TIMESTEP;
        if TIMESEC>=60 then
          begin
        TIMEMIN:=TIMEMIN+(TIMESEC)div(60);TIMESEC:=(TIMESEC)mod(60)
          end;
        if TIMEMIN>=60 then
          begin
        TIMEHR:=TIMEHR+(TIMEMIN )div (60);
        TIMEMIN:=TIMEMIN mod 60;
          end;
        if TIMEHR>=24 then TIMEHR:=TIMEHR-24;
          end;
        WRITELN(TTY,'TIME:':50,TIMEHR:2,':',TIMEMIN:2,':',TIMESEC:2)
  end;
    %*****************************************************************************\
  procedure GETDIRECTION(DIR:RANGE;var DIRWORD:LETTERWORD);
  (* converts DIRECTION to the cardinal direction .
used for outputting -like flow of river *)
    %************************************************************************\
  begin
      case DIR of
      0:DIRWORD:='NORTH      ';1:DIRWORD:='NORTH EAST  ';
      2:DIRWORD:='EAST       ';3:DIRWORD:='SOUTH EAST  ';
      4:DIRWORD:='SOUTH      ';5:DIRWORD:='SOUTH WEST  ';
      6:DIRWORD:='WEST       ';7:DIRWORD:='NORTH WEST  ';
      others:
      end
  end;
    %*****************************************************************************\
  procedure REMOVESHOCK;
  (* resets the tank to resume its activity after it has been hit
and SHOCKED out of action *)
  var
      I,J:integer;
  begin
      for I:=1 to 5 do
      for J:=1 to 3 do
      with TNKDAT[I,J].STATUS do
             if SHOCKED then SHOCKED:=false
  end;
    %*****************************************************************************\
  procedure GETDIRECTION(X1,X2,Y1,Y2:integer;var DIRECTION:integer);
  %*****************************************************************************\
  var
      XDISP,YDISP:integer;ANGLE:real;
      %*****************************************************\
  procedure CALLANGLEX(BASE:integer);
  %*****************************************************\
        begin
      if ANGLE>=2.35 then DIRECTION:=0
      else
      if ANGLE >= 0.42 then DIRECTION:=ABS(1-BASE)
      else DIRECTION:=ABS(2-BASE)
        end;
```

```
     %****************************************************\
procedure CALLANGLEY(BASE:integer);
%****************************************************\
     begin
   if ANGLE <= 0.42 then DIRECTION:=ABS(BASE-2)
   else
   if ANGLE <= 2.35 then DIRECTION:=ABS(BASE-3)
   else DIRECTION:=4
     end;
begin
   XDISP:=X2-X1;
   YDISP:=Y2-Y1;
   if XDISP=0 then
     begin
   if YDISP > 0 then DIRECTION:=0
   else DIRECTION:=4
     end
   else
     begin
   ANGLE:=ABS(YDISP/XDISP);
   if XDISP >0 then
     begin
   if YDISP>0 then CALLANGLEX(0)
   else CALLANGLEY(0)
     end
   else
   if YDISP>0 then CALLANGLEX(8)
   else CALLANGLEY(8)
     end
end;
     %***************************************************************************\
procedure FILLTOPOGRaphy;
(* takes the details described earlier from file MAPDAT and stores it for use*)
(* when this is completed a note to this effect is given on TTY *)
%*****************************************************************************\
type
   SPLITARRAY=packed array[1..2]of 0..99;
var
   WIDTH,HEIGHT,DEPTH,I,DEGREE:integer;
   CUR1,CUR2:integer;
   FIXEDWORD:packed array[1..10]of char;
   DIRECTION:0..8;NEXTPOINTER,LASTPOINTER,CURRENTPOinter:SPLITARRAY;
   FIRSTPOINT,ERROR:boolean;
   %*************************************************************************\
procedure GETPTR;
%*****************************************************************\
     begin
   NEW(PTR);PTR^.NEXTPTR:=nil;
   PTR^.VALUE1:=0;
   PTR^.VALUE2:=0
     end;
   %*************************************************************\
procedure  FILLGRIDChar(P1,P2:integer);
%*****************************************************************\
     begin
   GETPTR;
```

```
08410           GRIDCHAR[P1,P2].GROUNDPTR:=PTR;
08420           GRIDCHAR[P1,P2].TYPOFGROUND:=I
08430             end;
08440           {*****************************************************\
08450         procedure COMBINETYPofground(P1,P2:integer);
08460         (*checks for ground when it has 2 or more caracteristics at one grid point *)
08470         {*****************************************************\
08480             var
08490           TEMPGROUND:1..9;
08500             begin
08510           TEMPGROUND:=GRIDCHAR[P1,P2].TYPOFGROUND;
08520           GRIDCHAR[P1,P2].TYPOFGROUND:=7;
08530           GETPTR;
08540           PTR^.NEXTPTR:=GRIDCHAR[P1,P2].GROUNDPTR;
08550           GRIDCHAR[P1,P2].GROUNDPTR:=PTR;
08560           PTR^.VALUE1:=TEMPGROUND;
08570           PTR^.VALUE2:=1;
08580           while PTR^.NEXTPTR<>nil do PTR:=PTR^.NEXTPTR;
08590           TEMPPTR:=PTR;GETPTR;
08600           TEMPPTR^.NEXTPTR:=PTR
08610             end;
08620           {*****************************************************\
08630         procedure FILLADVGRID;
08640         (* puts the  attribute of type of ground at the next grid point with direction *)
08650         {*****************************************************\
08660             begin
08670           if GRIDCHAR[CURRENTPOInter[1],CURRENTPOInter[2]].TYPOFGROUND=8 then
08680           FILLGRIDChar(CURRENTPOInter[1],CURRENTPOInter[2])
08690           else COMBINETYPofground(CURRENTPOInter[1],CURRENTPOInter[2]);
08700           PTR^.VALUE1:=(DIRECTION+4)mod(8)
08710             end;
08720           {*****************************************************\
08730         procedure FILLPRESENTgrid;
08740         (* puts the attribute of type of ground at the present point with direction *)
08750         {*****************************************************\
08760             begin
08770           if FIRSTPOINTER then
08780           if GRIDCHAR[LASTPOINTEr[1],LASTPOINTEr[2]].TYPOFGROUND<>8 then
08790           COMBINETYPofground(LASTPOINTEr[1],LASTPOINTEr[2])
08800           else FILLGRIDChar(LASTPOINTEr[1],LASTPOINTEr[2])
08810           else
08820             begin
08830           if GRIDCHAR[LASTPOINTEr[1],LASTPOINTEr[2]].TYPOFGROUND=1 then
08840           PTR:=GRIDCHAR[LASTPOINTEr[1],LASTPOINTEr[2]].GROUNDPTR
08850           else
08860             begin
08870                 PTR:=GRIDCHAR[LASTPOINTEr[1],LASTPOINTEr[2]].GROUNDPTR;
08880                 while PTR^.NEXTPTR<>nil do PTR:=PTR^.NEXTPTR
08890             end;
08900             end;
08910           PTR^.VALUE2:=DIRECTION;
08920           if I<3 then
08930             begin
08940           TEMPPTR:=PTR;GETPTR;
08950           PTR^.VALUE1:=FIDF4;
08960           PTR^.VALUE2:=DSPTH;TEMPPTR^.NEXTPTR:=PTR
```

```
08970             end
08980             end;
08990           %*************************************************************\
09000         procedure READCHAR;(*READS A CHARACTER FROM THE FILE MAPDAT *)
09010         %*************************************************************\
09020             var
09030           ALP:char;
09040             begin
09050           repeat READ(MAPDAT,CH)
09060           until CH#' ';
09070           while not (EOLN(MAPDAT)or(ALP='.')) do READ(MAPDAT,ALP);
09080             end;
09090           %*************************************************************\
09100         procedure CONVERTDEGree(var DIRECTION:RANGE);
09110         (* converts DEGREE to direction and outputs it in DIRECTION *)
09120         %*************************************************************\
09130             begin
09140           case DEGREE OF
09150           0:DIRECTION:=0;45:DIRECTION:=1;
09160           90:DIRECTION:=2;135:DIRECTION:=3;
09170           180:DIRECTION:=4;225:DIRECTION:=5;
09180           270:DIRECTION:=6;315:DIRECTION:=7
09190           end;
09200             end;
09210           %*************************************************************\
09220         procedure READINCHAR;(*READS DIRECTLY INPUT UPTO 12 CHARACTERS FROM FILE MAPDAT*)
09230         %*************************************************************\
09240             var
09250           INCH:char;I:integer;
09260             begin
09270           for I:=1 to 12 do WORD[I]:=' ';
09280           INCH:=' ';I:=1;
09290           while INCH=' ' do READ(MAPDAT,INCH);
09300           while ((not EOLN(MAPDAT))and(I<13)and(INCH#',')) do
09310             begin
09320           WORD[I]:=INCH;I:=I+1;READ(MAPDAT,INCH);
09330             end;
09340           if I>12 then ERR(3)
09350           else
09360           if EOLN(MAPDAT)then WORD[I]:=INCH
09370             end;
09380           %*************************************************************\
09390         procedure READNUMBER;
09400         %*************************************************************\
09410             begin
09420           READINCHAR;
09430           NUMBER
09440             end;
09450           %*************************************************************\
09460         procedure SPLIT(var POINT:SPLITARRAY);
09470         %*************************************************************\
09480             begin
09490           POINT[1]:=INTVAL div 100;
09500           POINT[2]:=INTVAL mod 100
09510             end;
09520           %*************************************************************\
```

```
09530       procedure FILLGRID;
09540       (* completes the grid heights after the contours have been described *)
09550           type
09560         GRIDRECORD=record
09570               HT,NO:integer
09580             end;
09590           var
09600         STACK:array[1..100] of GRIDRECORD;
09610         TOP,I,J:integer;
09620         {*****************************************************************\
09630           procedure FILLGRID(FROMNO,TONO,HT:integer);
09640           (*fills the heights HT between pts FROMNO to TONO *)
09650           {*****************************************************************\
09660           var
09670         K:integer;
09680           begin
09690         if TOP=0 then MSG(2)
09700         else
09710           begin
09720               for K:=FROMNO to TONO do
09730               if GRIDCHAR[K,I].ALTITUDE=0 then
09740               GRIDCHAR[K,I].ALTITUDE:=HT;
09750               STACK[TOP].HT:=0;STACK[TOP].NO:=0;TOP:=TOP-1
09760           end;
09770           end;
09780         {*****************************************************************\
09790           procedure PUSH(HEIGHT,NUMBER:integer);
09800           (*stores the height HEIGHT in stack for matching *)
09810           {*****************************************************************\
09820           var
09830         PUSHSTACK:boolean;
09840           begin
09850         PUSHSTACK:=false;
09860         if TOP<2 then PUSHSTACK:=true
09870         else
09880           begin
09890               if ((STACK[TOP].HT>STACK[TOP-1].HT)and(HEIGHT>STACK[TOP].HT))
09900               then PUSHSTACK:=true
09910               else
09920               if ((STACK[TOP].HT<STACK[TOP-1].HT)and(HEIGHT<STACK[TOP].HT))
09930               then  PUSHSTACK:=
09940               true
09950               else
09960               if HEIGHT=STACK[TOP-1].HT then
09970           begin FILLGRID(STACK[TOP-1].NO,NUMBER,HEIGHT);STACK[TOP].HT:=0;
09980               STACK[TOP]
09990               .NO:=0;TOP:=TOP-1;
10000           end
10010               else MSG(1);
10020           end;
10030         if PUSHSTACK then
10040           begin
10050               TOP:=TOP+1;
10060               if TOP >10 then MSG(1)
10070               else
10080           begin
```

```
            with STACK[TOP] do
              begin
              HT:=HEIGHT;WD:=NUMBER;
                end;
        end;
        end;
        end;
      {***********************************************************************\
        procedure INITSTACK;
        (* initialises the stack for starting *)
        {***********************************************************************\
        var
      TS:integer;
        begin
      for TS:=0 to 10 do with STACK[TS] do
        begin
            HT:=0;WD:=0
        end;
      TOP:=0;
        end;
        begin
      for I:=1 to 40 do
        begin
      INITSTACK;
      for J:=1 to 40 do
        begin
            if GRIDCHAR[J,I].ALTITUDE#0 then
        begin
            if STACK[TOP].HT#GRIDCHAR[J,I].ALTITUDE then
            PUSH(GRIDCHAR[J,I].ALTITUDE,J)
            else FILLGRID(STACK[TOP].WD,J,STACK[TOP].HT)
        end;
        end;
        end;
        end;
  begin
      for I:=1 to 7 do
        begin
      case I of
      1:FIXEDWORD:='RIVER      ';
      2:FIXEDWORD:='CANAL      ';
      3:FIXEDWORD:='ROAD       ';
      4:FIXEDWORD:='CROP/FEILD';
      5:FIXEDWORD:='MINE FEILD';
      6:FIXEDWORD:='MARSHY GD ';
      7:FIXEDWORD:='CONTOUR    ';
      end;
      loop
      READCHAR;ERROR:=false;
      exit if CH#'Y';
      if I<3 then
        begin
            READ(MAPDAT,WIDTH);
            READ(MAPDAT,DEPTH);
        end
      else
```

```
10650           if I=7 then
10660             begin
10670               READ(MAPDAT,HEIGHT)
10680             end;
10690           READNUMBER;FIRSTPOINT:=true;
10700           SPLIT(LASTPOINTER);
10710           loop
10720               READNUMBER;
10730           exit if ((CHR='FINISH      ')or(not ISNUMBER)or(ERROR));
10740                 SPLIT(NEXTPOINTER);
10750                 GETDIRECTION(LASTPOINTER[1],NEXTPOINTER[1],LASTPOINTER[2],
10760                 NEXTPOINTER[2],DIRECTION);
10770                 repeat
10780                 CURRENTPOInter:=LASTPOINTER;
10790                 CUR1:=CURRENTPOInter[1];CUR2:=CURRENTPOInter[2];
10800                 CALCNEXTGRID(CUR1,CUR2,DIRECTION);
10810                 CURRENTPOInter[1]:=CUR1;CURRENTPOInter[2]:=CUR2;
10820                 if ((CURRENTPOInter[1]<41)and(CURRENTPOInter[1]>0)
10830                 and(CURRENTPOInter[ 2]<41)and(CURRENTPOInter[2]>0)) then
10840                   begin
10850                 if I<7 then
10860                   begin
10870                 FILLADVGRID;FILLPRESENTGRID;
10880                   end
10890                 else GRIDCHAR[LASTPOINTER[1],LASTPOINTER[2]],ALTITUDE
10900                 :=HEIGHT;
10910                 LASTPOINTER:=CURRENTPOInter;FIRSTPOINT:=false;
10920                   end
10930                 else
10940                   begin
10950                 MSG(11);
10960                 ERROR:=true;
10970                   end
10980                 until
10990                 ((CURRENTPOInter=NEXTPOINTER)or (ERROR));
11000                 if CURRENTPOInter*NEXTPOINTER then MSG(12);
11010           end;
11020           end;
11030             end;
11040           FILLALTITUde;
11050           WRITEGRID('IMPORTANT:note that the map details have been taken
11060     from your file = "MAPDAT" =');
11070           SKIP(2);
11080       end;
11090         {*****************************************************************\
11100       function ISVISIBLE(LOC1X,LOC1Y,LOC2X,LOC2Y:integer;C":char):boolean;
11110       (* checks if the loc1 and loc2 are visible taking into consideration all other fact
11120        except that of line of sight *)
11130       {*****************************************************************\
11140       label
11150           10;
11160       var
11170           FINISH,RIGHT:boolean;
11180           DISTANCE,FIVE:integer;
11190           {*****************************************************************\
11200       function CHECKVISIBility(LOC1X,LOC1Y,LOC2X,LOC2Y:integer):boolean;
```

```
11210        (* checks if the line of sight visibility is there or not *)
11220        %*************************************************************\
11230          var
11240        CHECKX,LOC1:boolean;
11250        INCR,STARTX,STARTY,ENDX,ENDY,CURRENTX,CURRENTY:integer;
11260        CURRANGLE,ANGLE,M,C:real;
11270        %*************************************************************\
11280          procedure CALNEXTX;
11290          %***********************************************************\
11300          begin
11310        CURRENTX:=CURRENTX + INCR;
11320        CURRENTY:=TRUNC(M*CURRENTX+C+0.5)
11330          end;
11340        %*************************************************************\
11350          procedure CALNEXTY;
11360          %***********************************************************\
11370          begin
11380        CURRENTY:=CURRENTY + INCR;
11390        CURRENTX:=TRUNC(((CURRENTY-C)/M)+0.5)
11400          end;
11410          begin
11420        if ABS(LOC1X-LOC2X) > ABS(LOC1Y-LOC2Y) then CHECKX:=true
11430        else CHECKX:=false;
11440        M:=(LOC2Y-LOC1Y)/(LOC2X-LOC1X);
11450        C:=LOC1Y-M*LOC1X;
11460        if GRIDCHAR[LOC1X,LOC1Y].ALTITUDE > GRIDCHAR[LOC2X,LOC2Y].ALTITUDE then
11470        LOC1:=true
11480        else
11490        LOC1:=false;
11500        CALCULATEDist(LOC1X,LOC1Y,LOC2X,LOC2Y,DISTANCE);
11510        if LOC1 = true then
11520          begin
11530        STARTX:=LOC2X;STARTY:=LOC2Y;
11540        ENDX:=LOC1X;ENDY:=LOC1Y;
11550          end
11560        else
11570          begin
11580        STARTX:=LOC1X;STARTY:=LOC1Y;
11590        ENDX:=LOC2X;ENDY:=LOC2Y;
11600          end;
11610        CURRENTX:=STARTX;CURRENTY:=STARTY;
11620        ANGLE:=(GRIDCHAR[STARTX,STARTY].ALTITUDE - GRIDCHAR[ENDX,ENDY].ALTITUDE)/
11630        DISTANCE;
11640        if CHECKX then
11650        if ENDX > STARTX then INCR:=1
11660        else INCR:=-1
11670        else
11680        if ENDY > STARTY then INCR:=1
11690        else INCR:=-1;
11700        CURRANGLE:=0.0;
11710        while (CURRANGLE<ANGLE)and((CURRENTX<>ENDX)or(CURRENTY<>ENDY))do
11720          begin
11730        if CHECKX then CALNEXTX
11740        else CALNEXTY;
11750        CALCULATEDist(CURRENTX,CURRENTY,ENDX,ENDY,DISTANCE);
11760        CURRANGLE:=(GRIDCHAR[CURRENTX,CURRENTY].ALTITUDE - GRIDCHAR[ENDX,ENDY].
```

```
          end
        else
          begin
              ERROR(1);
              ERR:=true;
          end
          end
          end
        else
          begin
        READWORD;NUMBER;
        if ISNUMBER then
          begin
        LOC2X:=TNKDAT[NUMWORD[1],NUMWORD[2]].MAPCORD.X;
        LOC2Y:=TNKDAT[NUMWORD[1],NUMWORD[2]].MAPCORD.Y
          end
        else ERR:=true
          end
        ;
        if not ERR then
          begin
        case CH of
        'L':WRITELN(TTY,'LANDMARK',TEMP,'LOCATION',LOC2X,LOC2Y);
        'T':WRITELN(TTY,'TANK','LOCATION',LOC2X,LOC2Y);
        others:
        end;
        if ISVISIBLE(LOC1X,LOC1Y,LOC2X,LOC2Y,CH) then WRITELN(TTY,'IS VISIBLE TO ME')
        else WRITELN(TTY,'IS NOT VISIBLE TO ME');
          end;
  end;
    %*****************************************************************************
  procedure REPORT( CH:char);(*REPORTS ACTIVITY OF ENEMY OR LOCATION OR FUEL BALANCE*)
  (* output details like FUEL BALANCE,LOCATION taking the
details from the record TNKDAT for the tank IK,JK *)
  %*****************************************************************************\
  begin
      WRITELN(TTY,'TANK ',JK:2,' OF TROOP ',IK:2,' REPORTING':12);WRITELN(TTY);
      if CH='F' then
        begin WRITE(TTY,'FUEL BALANCE:':25);WRITELN(TTY,TNKDAT[IK,JK].FUELCAP:6,'  LI
      )
        end;
      if CH='L' then
        begin WRITE(TTY,'LOCATION':10);WRITELN(TTY,'X':10,'Y':10);
      CONVERTGRIdref(GRX,GRY,TNKDAT[IK,JK].MAPCORD.X,TNKDAT[IK,JK].MAPCORD.Y);
      WRITELN(TTY,GRX:21,GRY:10)
        end;
      if CH='E' then
        begin WRITELN(TTY,'ENEMY ACTIVITY');WRITELN(TTY,'ENEMY SEEN AT LOC 628105')
        end
  end;
    %*****************************************************************************
  procedure   REPORTAMMUnation;(*REPORTS BALANCE OF AMMUNITION OF ALL TYPES*)
  (* selects the type of ammunation balance for the tank IK,JK and print the balance
  %*****************************************************************************\
  var
      AP,SM,H,ALL:boolean;I,J:integer
```

```
12890         ;
12900       begin
12910         READWORD;
12920         if WORD='                ' then
12930           begin
12940         AP:=false;H:=false;SM:=false;
12950         repeat
12960         if WORD='AP              ' then
12970           begin AP:=true;READWORD
12980           end;
12990         if WORD='HE              ' then
13000           begin H:=true;READWORD
13010           end;
13020         if WORD='SMOKE           ' then
13030           begin  SM:=true;READWORD
13040           end;
13050         if WORD='ALL             ' then
13060           begin ALL:=true;READWORD
13070           end
13080         until (WORD<>'AP              ')or(WORD<>'SMOKE           ')or(WORD<>'HE              ');
13090           end
13100         else  ALL:=true;
13110         WRITELN(TTY,'TANK':10,IK:6,'OF TROOP':10,JK:6,'REPORTING AMMUNATION BALANCE':32);
13120         WRITELN(TTY,'TYPE':20,'QUANTITY':20);
13130         if ((AP=true)or(ALL=true)) then WRITELN(TTY,'AP':19,TNKDAT[IK,JK1,AMNCAP,TYP1:2
13140         if (SM=true)or(ALL=true)
13150         then
13160         WRITELN(TTY,'SMOKE ':20,TNKDAT[IK,JK1,AMNCAP,TYP2:19);
13170         if (H=true)or(ALL=true) then WRITELN(TTY,'HE  ':20,TNKDAT[IK,JK1,AMNCAP,TYP3:19)
13180       end;
13190         %*************************************************************\
13200       procedure TANKSTATUS( TA:boolean);(*REPORTS STATUS OF OWN TANK OR OTHER TANK*)
13210       (* depending on the type of command reports the STATUS of self or of other tanks
13220        like BOGGED,MOVING etc *)
13230       %*************************************************************\
13240       var
13250         J,K,I:integer;TEMP:packed array[1..4]of char;LOCREC:boolean;
13260       begin LOCREC:=false;
13270         if TA=true then
13280           begin  READWORD;
13290         if WORD='LOC             ' then
13300           begin
13310         READWORD; for K:=1 to 4 do TEMP[K]:=WORD[K];
13320         LOCREC:=true
13330           end
13340         else
13350           begin
13360         NUMBER:J:=NUMWORD[2];I:=NUMWORD[1];READWORD;
13370         if WORD='LOC             ' then
13380           begin READWORD;for K:=1 to 6 do TEMP[K]:=WORD[K];
13390             LOCREC:=true
13400           end
13410           end
13420           end;
13430         WRITELN(TTY,'TANK':10,IK:3,'OF TROOP':10,JK:3);
13440         if TA=true then
```

```
13450             begin IK:=I; IK:=J;
13460           WRITELN(TTY,'REPORTING STATUS OF TANK':30,I:3,' OF TROOP':10,J:3);
13470           if LOCSEC=true then WRITELN(TTY,'located at grid ref  ',MAPCORD);
13480             end
13490           else
13500           WRITELN(TTY,'REPORTING STATUS OF SELF':30);
13510           with TROOP[IK,JK] do with STATUS do
13520             begin
13530           SKIP(2);
13540           if DAMAGED then MSG(4);
13550           if OUTOF then MSG(5)
13560           else WRITELN(TTY,'STANDING AT LOC ',MAPCORD.X,MAPCORD.Y);
13570           if FLAGS then MSG(6);
13580           if not CANSHOOT then MSG(7);
13590           if not CANMOV then
13600             begin
13610                 MSG(8);
13620                 if SHOCKED then MSG(9)
13630                 else MSG(10)
13640             end
13650             end;
13660           SKIP(2)
13670       end;
13680           {*****************************************************************\
13690       procedure   REPORTPROC;(*ANALYISES THE REPORT PROCEDURE*)
13700       (* analyses the REPORT procedure and decides the type of report that is required *
13710       {*****************************************************************\
13720       begin READWORD;
13730           if WORD='STATUS        ' then
13740             begin READWORD;
13750           if WORD='.             ' then TANKSTATUS(false)
13760           else
13770           if WORD='IK            ' then TANKSTATUS(true)
13780           else ERROR(9)
13790             end
13800           else
13810             begin
13820           if WORD='LOCATION      ' then REPORT('L')
13830           else
13840           if WORD='BALANCE       ' then
13850             begin READWORD;
13860           if WORD='AMMUNATION    ' then REPORTAMMUnation
13870           else
13880           if WORD='FUEL          ' then REPORT('F')
13890           else ERROR(8)
13900             end
13910           else
13920           if WORD='ENEMY         ' then REPORT('E')
13930           else
13940           if WORD='VISIBILITY    ' then
13950             begin READWORD;
13960           if WORD='UM            ' then REPORTVISIbility('U')
13970           else
13980           if WORD='IK            ' then REPORTVISIbility('I')
13990           else ERROR(10)
14000             end
```

```
          else ERROR(3)
            end
      end;
          %*****************************************************************************\
      procedure CALCULATEPos(A,B:integer);
    (*CALCULATES NEW POSITION WHEN THE TANK IS ON THE MOVE*)
      (* this is used  to calculate destination for MOVE
    when the move command specifies only DIRECTION and DISTANCE *)
      (* A and B are the present position. DEG gives the direction
    of move in degrees and DIST gives   the  distance to be   moved *)
      (* the destination cordinates are updated  through MQ^.DEST *)
      %****************************************************************************\
      begin
          ANGLE:=(DEG/180)*3.14;
          MQ^.DEST^.X:=A+TRUNC(DIST*SIN(ANGLE)/500);
          MQ^.DEST^.Y:=B+TRUNC(DIST*COS(ANGLE)/500);
      end;
          %*****************************************************************************\
      procedure CHECKDIRECtion(var DEG:integer);(*CONVERTS CARDINAL DIRECTIONS TO DEGREES
      %****************************************************************************\
      begin
          DEG:=400;
          if((WORD='NORTH        ')or(WORD='N          '))then DEG:=0;
          if((WORD='NORTH-EAST   ')or(WORD='NE         '))then DEG:=45;
          if((WORD='EAST         ')or(WORD='E          ')) then DEG:=90;
          if((WORD='SOUTH-EAST   ')or(WORD='SE         ')) then DEG:=135;
          if((WORD='SOUTH        ')or(WORD='S          ')) then DEG:=180;
          if((WORD='SOUTH-WEST   ')or(WORD='SW         ')) then DEG:=225;
          if((WORD='WEST         ')or(WORD='W          ')) then DEG:=270;
          if((WORD='NORTH-WEST   ')or(WORD='NW         ')) then DEG:=315
      end;
          %*****************************************************************************\
      procedure   FILLRESTMOvtab;(*COMPLETES THE MOVE TABLE*)
      (* copies the rest of the data  of tank from TNKDAT after
    checking its validity   of MOVE command. *)
      (* the return of ROGER ! segnifies that this tank is added to the MOVE list *)
      %****************************************************************************\
      begin
          MQ^.SOURCE.X:=TNKDAT[IK,JK].MAPCORD.X;
          MQ^.SOURCE.Y:=TNKDAT[IK,JK].MAPCORD.Y;
          MQ^.LOC.X:=TNKDAT[IK,JK].MAPCORD.X;
          MQ^.LOC.Y:=TNKDAT[IK,JK].MAPCORD.Y;
          MQ^.STATE:=true;MLAST^.LLINK:=MQ;
          TNKDAT[IK,JK].STATUS.MOVING:=true;
          MQ^.CURROBJECTive:=1;
          MQ^.ULINK:=MLAST;MLAST:=MQ;WRITELN(TTY,'ROGER !');SKIP(1)
      end;
          %*****************************************************************************\
      procedure CHECKLANDmark;(*CHECKS IF THE LANDMARK EXISTS IN THE TABLE OF LANDMARKS*)
      (* checks LANDMARKS so that the same name landmarks are not redeclared.
       (* it checks if the landmark exist and returns its pointer in GM and sets
       ISLANDMARK as true   *)
      %****************************************************************************\
      begin
          ISLANDMARK:=false;GM:=0;
          if LMROOT^.RLINK#nil then
```

```
          begin
          LP:=LANDPTR^.R,LINK;
          while (LP^.NAME<>WORD)and(LP#nil) do
            begin
          if WORD>LP^.NAME then LP:=LP^.RLINK
          else LP:=LP^.LLINK
            end;
          if LP^.NAME=WORD then
            begin
          ISLANDMARK:=true;GR:=LP^.GR
            end;

            end
      end;
          (**************************************************************************\
      procedure CHECKGRIDREF(var A,B:integer);
      (* checks two things ,firstly if the WORD is a NUMBER or LANDMARK.if it is
       then converts and gets its griderf in A, B *)
          ***************************************************************************\
      var
          GR:integer;
      begin
          READWORD;
          if not ISNUMBER then CHECKLANDMark(GR);
          if ISNUMBER or ISLANDMARK then
            begin
          if ISNUMBER then CONVERT(INTVAL,A,B)
          else CONVERT(GR,A,B)
            end
      end;
          (***************************************************************************\
      procedure ENTERROUTE;(*ENTERS THE ROUTE TO BE TAKEN BY THE TANK ON THE MOVE*)
      (* this is part of the MOVE table filling. it fills the route to be taken
       by the tank if specified in input command. *)
      (* max points are limited to 10. the no of points are kept in MO^,OBJECTIVE *)
      (* the current point is used at he time of processing and
    is stored in MO^.CURROBJECTIVE *)
          ***************************************************************************\
      var
          J,I,GR:integer;A,B:integer;
      begin
          I:=0;for J:=1 to 10 do
            begin
          MO^.ROUTE[J].X:=0;
          MO^.ROUTE[J].Y:=0
            end;
          loop
          READWORD;
          exit if ((WORD='.              ')or(I=11));
          CHECKGRIDRef(A,B);
          if (ISNUMBER or ISLANDMARK)then
            begin
          I:=I+1;MO^.ROUTE[I].X:=A;MO^.ROUTE[I].Y:=B
            end
          else ERROR(18)
          end;
```

```
        if not ISL[illegible] then FILLRESTMOVtab
    end;
    (**************************************************************************)
    procedure ENTERDISTANCE/tab;(*ANALYSES THE DISTANCE OF MOVE COMMAND*)
    (* this processes one branch of the move command where DIRECTION and DISTANCE
     is used to describe new location *)
    (**************************************************************************)
    var
       [illegible],A,B:integer;
    begin
        NUMBER;DEG:=400;
        if ISNUMBER then
          begin
        DIST:=INTVAL;READWORD;
        if ((WORD='METERS       ')or(WORD='MTRS         ')or(WORD='MTR          ')
              or(WORD='KM           ')or( WORD='KMS          '))
        then
          begin
        if ( (WORD='KM           ') or ( WORD='KMS          '))
        then DIST:=DIST*1000;
        READWORD;NUMBER;
        if((ISNUMBER)and(INTVAL<361)) then
          begin
              DEG:=INTVAL;READWORD
          end
        else
        if not ISNUMBER then CHECKDIRECTion(DEG)
        else ERROR(13);
        if DEG<361 then
          begin
              READWORD;
              if WORD='.            ' then
          begin
              CALCULATEPosn(TNKDAT[IK,JK].MAPCORD.X,
              TNKDAT[IK,JK].MAPCORD.Y);
              FILLRESTMOVtab
          end
              else
              if WORD='OF           ' then
          begin READWORD; CHECKGRIDREF(A,B);
              if  ISNUMBER or ISLANDMARK then
                begin

              CALCULATEPosn(A,B);READWORD;
              if

              WORD='ROUTE        ' then ENTERROUTE
              else FILLRESTMOVtab
                end
              else ERROR(14)
          end
              else
              ERROR(15)
          end
        else ERROR(16)
```

```
            end
          else ERROR(17)
            end;
    end;
          %***********************************************************************\
    procedure [illegible];(*ANALYISES THE OBJECTIVE PART OF THE MOVE [illegible]*)
    (* this processes the branch of MOVE where a straight
 forward move to objective is there *)
    %*************************************************************************\
    var
        [illegible]:integer;[illegible]:integer;
    begin
        [illegible]ef(A,B);
        if IS[illegible] or ISLANDMARK then
          begin
        [illegible]^.DEST.X:=A;[illegible]^.DEST.Y:=B;READWORD;
        if [illegible]='.                    ' then FILLRESTMOVtab
        else
        if [illegible]='[illegible]          ' then ENTERROUTE
        else ERROR(2?)
          end
        else ERROR(21)
    end;
          %***********************************************************************\
    procedure [illegible]ab(TEMPPTR:MPTR);(*REMOVES ATANK FROM THE MOVE TABLE*)
    (* this deletes an entry in movetable addressed by TEMPPTR . used at reaching
     of destination or nuetralisation or re-entering of table *)
    %*************************************************************************\
    var
        TEMP:MPTR;
    begin
        TEMP:=TEMPPTR^.ULINK;TEMP^.LLINK:=TEMPPTR^.LLINK;
        TEMP:=TEMPPTR^.LLINK;
        if TEMP<>nil then TEMP^.ULINK:=TEMPPTR^.ULINK
        else [illegible]:=TEMPPTR^.ULINK
    end;
          %***********************************************************************\
    procedure CHECKMOVTAB;(*CHECKS IF THE TANK IS ALREADY IN THE [illegible]*)
    %*************************************************************************\
    begin
        ISTHMOVTAB:=false;
        MR:=MOVTABROOT;while ((MR^.TANKNO<>TANKNUMBER)and(MR<>[illegible]LAST)) do
        MR:=MR^.LLINK;
        if MR^.TANKNO=TANKNUMBER then ISTHMOVTAB:=true
    end;
          %***********************************************************************\
    procedure MOVEPROCEDure;
    %*************************************************************************\
        forward;
          %***********************************************************************\
    procedure CHANGEDESTination;(*CHANGE DESTINATION WHEN THE TANK IS IN THE MOVE[illegible])
    %*************************************************************************\
    var
        A,B:integer;
    begin
        WRITE(TTY,'NEW DESTINATION:');BREAK;
```

```
           READINPUT;
           CHECKGRIDREF(?,?);
           if ISNUMBER or ISLANDMARK then
             begin
             MO^.DEST.X:=[illegible];MO^.STATE:=true;
             MO^.DEST.Y:=[illegible]
             end
           else
             begin
             ERROR(71);
             CHANGEDESTINATION
             end
   end;
   %*********************************************************************\
   procedure CHANGEROUTE;(*CHANGES ROUTE OF A TANK ON MOVE*)
   %*********************************************************************\
   begin
       WRITE(TTY,'NEW ROUTE:');BREAK;
       READLINE;MO:=MR;
       MO^.STATE:=true;
       ENTERROUTE
   end;
   %*********************************************************************\
   procedure MOVTABCHANGE;(*CHANES THE MOVE TABLE*)
   %*********************************************************************\
   begin
       DELETEMOVTAB(MR);
       WRITE(TTY,'CHANGE TO-MOVE:');BREAK;READLINE;
       MOVEPROCEDURE
   end;
   %*********************************************************************\
   procedure ALTERMOVTAB;(*CHECKS THE PART TO BE CHANGED IN MOVE TABLE*)
   %*********************************************************************\
   begin
       WRITE(TTY,'TANK ALREADY ON MOVE. DO YOU WANT TO CHANGE STATUS ?');BREAK;
       READCH;
       if CH='Y' then
         begin
       WRITELN(TTY,'FOR CHANGING ONLY DESTINATION TYPE "D"');
       WRITELN(TTY,'FOR CHANGING ONLY ROUTE TYPE "R"');
       WRITELN(TTY,'FOR COMPLETE CHANGING "C"');
       WRITELN(TTY,'FOR RETURN WITHOUT CHANGE "N"');
       WRITE(TTY,'TYPE OF CHANGE REQUIRED ?');
       BREAK;READCH;
       case CH of
       'D':CHANGEDESTination;
       'R':CHANGEROUTE;
       'C':MOVTABCHANGE;
       'N':;
       others:MSG(2)
       end
         end
   end;
   %*********************************************************************\
   procedure MOVEPROCEDure;(*RESUMES MOVE OF A TANK*)
   %*********************************************************************\
```

```
begin
    CHECKMOVTAB;READ...;
    if IS...MOVTAB then
      begin
    if ...='.            ' then
      begin
    if MR^.STATE then MSG(1);
    MR^.STATE:=true
      end
    else ALTERSTEP
      end
    else
      begin
    ...(MQ);MQ^.ULINK:=nil;MQ^.TANKNO:=TANKNUMBER;MQ^.OBJECTIVE:=...;
    if ...='IO        ' then ENTERMOVTDobj
    else ENTERDISMOVtab
      end
end;
        %*******************************************************************\
procedure PRINTMOVTAB;(*PRINTS THE MOVE TABLE*)
%*******************************************************************\
begin
    WRITELN(TTY,'TANK NO':12,'DESTINATION':20,'STATE':15,'OBJECTIVE':13);
    MR:=MOVTABROOT;
    while MR<>MLAST do
      begin
    MR:=MR^.ULINK;
    WRITELN(TTY,MR^.TANKNO:12,MR^.DEST.X:10,MR^.DEST.Y,MR^.STATE:12,
       MR^.OBJECTIVE:10)
      end
end;
        %*******************************************************************\
procedure DELETEFTAB(FTEMP:FPTR);(*deletes a record with ptr FTEMP *)
%*******************************************************************\
begin
    if FTEMP=FLAST then
      begin  FLAST:=FTEMP^.ULINK;
    FTEMP^.ULINK^.ULINK:=nil;
      end
    else
      begin
    FTEMP^.ULINK^.ULINK:=FTEMP^.ULINK;
    FTEMP^.ULINK^.ULINK:=FTEMP^.ULINK;
      end;
end;
        %*******************************************************************\
function FIRETABPTR(I,J:integer):FPTR;
(* checks if the tank I,J is already firing , if it is then
returns its ptr value FPTR *)
%*******************************************************************\
var
    FTEMP:FPTR;
begin FTEMP:=FROOT;FIRETABPTR:=nil;
    while FTEMP<>FLAST do
      begin FTEMP:=FTEMP^.ULINK;
    if ((FTEMP^.SOURCETK[1]=I)and(FTEMP^.SOURCETK[2]=J))then
```

```
          begin
       WRITELN(TTY,'THIS TANK ALREADY FIRING,BALANCE OF',FIRE^.[illegible],
          'POUNDS LEFT TO FIRE');
       WRITELN(TTY,'DO YOU WANT TO CHANGE TARGET ?:');
       if YES then REASSIGN(FTEMP)
       else FIRINGSHELLS:=[illegible];
          end
          end;
    end;
       %*****************************************************************\
    procedure FIREPROCEDURE;
    (*FIRES the shells that are required to be fired pe tanks and checks its effect *)
    %*****************************************************************\
    label
       15;
    var
       Y,Z:integer;TEMP:LETTERWORD;
       %*****************************************************************\
    function PROBABILITY(I,J:integer):real;
    (* calculates the probability of hit depending on move etc of the tank *)
    %*****************************************************************\
          var
       VALUE:real;
          begin
       with TNKDAT[I,J].STATUS do
          begin
       PROBABILITY:=0.98;VALUE:=1.0;
       if BOGGED then
          begin VALUE:=0.8;PROBABILITY:=0.8;
          end;
       if MOVING then
          begin VALUE:=0.88;PROBABILITY:=0.88;
          end;
       if FIRING then PROBABILITY:=0.97*VALUE;
       if CUPOLACLOSed then PROBABILITY:=0.9*VALUE;
          end
          end;
       %*****************************************************************\
    procedure JOINFIRETAB;
    (*COPIES the rest of the details and joins the record to the table after checking *)
    %*****************************************************************\
          begin
       FLAST^.LLINK:=FP;FP^.ULINK:=FLAST;
       FLAST:=FP;
       if FP^.DESTTK[1]#[illegible] then
       FP^.HITPROB:=PROBABILITY(IK,JK)*PROBABILITY(FP^.DESTTK[1],FP^.DESTTK[2])
       else FP^.HITPROB:=PROBABILITY(IK,JK);
       FP^.TIME:=TNKDAT[IK,JK].LDTIME;
          end;
       %*****************************************************************\
    procedure GETNEWNODE;
    %*****************************************************************\
          var
       I:integer;
          begin
       NEW(FP);
```

```
17930            FP^.SOURCETK[1]:=IK;
17940            FP^.SOURCETK[2]:=JK;
17950            FP^.SOURCELOC.X:=[illegible][IK,JK].MAPCORD.X;
17960            FP^.SOURCELOC.Y:=[illegible][IK,JK].MAPCORD.Y;
17970            if [illegible].STATUS.MOVING then FP^.RATE:=GUNCHAR[TYPOFTANK].RATE[illegible]
17980            else FP^.RATE:=GUNCHAR[TYPOFTANK].RATEHALT;
17990            for I:=1 to 2 do FP^.DESTTK[I]:=0;
18000            FP^.[illegible]:=nil;
18010              end;
18020            {*****************************************************************\
18030          procedure GETLOCATION;
18040          (* gets the location of the tank in gridref *)
18050          {*****************************************************************\
18060              begin
18070            READWORD;
18080            TESTNUMBER:=true;NUMBER;
18090            if ISNUMBER then
18100              begin
18110            CONVERT(INTVAL,X,Y);
18120            FP^.DESTLOC.X:=X;FP^.DESTLOC,Y:=Y;
18130            JOIN:=true
18140              end
18150              end;
18160            {*****************************************************************\
18170          function CHECKOTHER(direction:boolean;
18180            {*****************************************************************\
18190              begin
18200            if ((WORD='BACK          ') or (WORD='FRONT         ') or (WORD='RIGHT         ')
18210                or (WORD='LEFT          ')) then
18220              begin READWORD;
18230            if WORD='OF            ' then
18240              begin
18250                READWORD;
18260                if WORD='YOU           ' then
18270                CHECKOTHER:=direction:=true;
18280              end;
18290              end
18300              end;
18310            end;
18320          begin
18330            if ((TNKDATA[IK,JK].STATUS.CANSHOOT)and(FIREIASPTR(IK,JK)=nil)) then
18340              begin
18350            JOIN:=false;
18360            TEMP:=WORD;TESTNUMBER:=true;
18370            READWORD;NUMBER;
18380            if ISNUMBER then
18390              begin
18400            GETNEWFNODE;FP^.TOFNUM:=INTVAL;
18410            READWORD;
18420            if WORD='SMOKE         ' then FP^.TYPOFAMM:=1
18430            else
18440            if WORD='HE            ' then FP^.TYPOFAMM:=2
18450            else
18460            if WORD='AP            ' then
18470              begin
18480                if TEMP='SHOOT         ' then MSG(1);
```

```
18490                 FP^.T(P)FA[ ]:=3
18500               end
18510             else
18520               begin ERROR(23);
18530                 goto 15
18540               end;
18550             READWORD;
18560             if (WORD='SHELLS      ') or (WORD='SHELLS      ') then
18570               begin
18580                 READWORD;
18590                 if WORD='A          ' then
18600                 begin
18610                   READWORD;
18620                   if ((WORD='OF          ')or(WORD='TANK        ')) then
18630                     begin
18640                     READWORD;NUMBER;
18650                     FP^.DESTK[1]:=NUMWORD[1];
18660                     FP^.DESTK[2]:=NUMWORD[2];
18670                     if ISNUMBER then
18680                       begin READWORD;
18690                       if WORD='ON          ' then GETLOCATION
18700                       else
18710                         begin
18720                           if ((not CHECKOTHERdirection)and(WORD#'.          ')) then
18730                           MSG(17);
18740                           FP^.DESTLOC.X:=TNKDAT[FP^.DESTK[1],FP^.
18750                           DESTK[2]].
18760                           MAPCORD.X;
18770                           FP^.DESTLOC.Y:=TNKDAT[FP^.DESTK[1],FP^.
18780                           DESTK[2]].
18790                           MAPCORD.Y;
18800                           JOIN:=true;
18810                         end;
18820                       end
18830                     else
18840                       begin
18850                       ERROR(24);
18860                       goto 15
18870                       end
18880                     end
18890                   else ERROR(25)
18900                 end
18910                 else
18920                 if WORD='ON          ' then GETLOCATION
18930                 else ERROR(26);
18940               end
18950             else ERROR(27)
18960               end
18970             else MSG(18)
18980               end;
18990             if JOIN=true then JOINFIREIAb;
19000         15:
19010         end;
19020           {*****************************************************************}
19030         procedure PROCESSMOVtank(PTR:MPIR);
19040         (* processses the move of the tank , checking for the obstackles and taking action
```

```
19050       %*****************************************************************\
19060       const
19070           GRADIENTFactor=1.45;
19080       var
19090           DIRECTION:RANGE;PERCENT,CURRENTTIME,NEXTX,NEXTY:integer;FINISH:boolean;
19100           SPEED,DISTANCE:integer; FACTOR:real;PERPAGE:boolean;
19110           %*****************************************************************\
19120       procedure CHANGEOFSECtion;
19130       (* gets the next point of objective once the grid pt has been reached *)
19140       %*****************************************************************\
19150           begin
19160           if PTR^.OBJECTIVE < PTR^.CURROBJECTive then
19170             begin
19180             NEXTX:=PTR^.OBST.X
19190             NEXTY:=PTR^.OBST.Y
19200             end
19210           else
19220             begin
19230             NEXTX:=PTR^.ROUTEPTR^.CURROBJECTive1.X;
19240             NEXTY:=PTR^.ROUTEPTR^.CURROBJECTive1.Y
19250             end
19260             end;
19270           %*****************************************************************\
19280       procedure COMPUTETIME;
19290       (* computes the time that the tank has taken to move from one pt to another *)
19300       %*****************************************************************\
19310           var
19320           X1,Y1,X2,Y2:integer;FACTOR:real;LPKM:1..5;
19330           TPTR:GRIDPTR;DISTANCE:integer;
19340           %*************************************************\
19350             procedure CHECKCROSSobstacle(GROUND:RANGE);
19360             (* checks for the obstacle crossing for ground type GROUND *)
19370             %*************************************************\
19380             begin LPKM:=5;
19390           if TANKCHART[TYPOFTANK].HEIGHT < TPTR^.NEXTPTR^.VALUE2 then
19400             begin MSG(12);MSG(15);REACHEDESTination:=true;
19410                 with TANKDATA[IK,JK].STATUS do
19420             begin
19430                 BOGGED:=true;CANMOV:=false;CANSHOOT:=false
19440             end
19450             end
19460           else
19470             begin
19480                 if ((GROUND=2)and(TANKCHART[TYPOFTANK].SPEEDTER <TPTR^.NEXTPTR^.VALUE2))
19490       then
19500             begin
19510                 MSG(12);MSG(15);FINISH:=true;
19520                 PTR^.STATE:=false
19530             end
19540                 else CROSSED:=true;
19550             end
19560             end;
19570           %*************************************************\
19580             procedure REPORTWATERobstacle(GROUND:RANGE);
19590             (* reports the encountering of obstacle and asks for the action to be taken*)
19600             %*************************************************\
```

```
   var
WORD:packed array[1..5]of char;
DIRWORD1,DIRWORD2:ALFA;
  begin
if GROUND=1 then WORD:='RIVER'
else WORD:='CANAL';
NUMTODIRECTion(TPTR^.VALUE1,DIRWORD1);NUMTODIRECTion(TPTR^.VALUE2,
                DIRWORD2);
WRITELN(TTY,'TANK ',IK:2,JK:2,'  REPORTING FROM LOCATION ',X1:3,Y1:3,' '
   :2,WORD,'  OBSTACLE AHEAD');
WRITELN(TTY,'WITH THE CHARACTERistics AS FOLLOWS:=');
SKIP(1);
WRITELN(TTY,'                         DIRECTION     ',DIRWORD1:12,'=>>     ',
   DIRWORD2:12);
WRITELN(TTY,'APPROX.WIDTH    ',TPTR^.NEXTPTR^.VALUE1:4,'    METERS');
WRITELN(TTY,'APPROX DEPTH    ',TPTR^.NEXTPTR^.VALUE2:4,'    CMS');
WRITELN(TTY,'SHOULD I NEGOTIATE THE OBSTACLE or CHANGE COURSE ?');
WRITE(TTY,'TYPE IN A "C" FOR CHANGING COURSE OR A "GO" TO CONTINUE:');
BREAK;READCH:=ER:=PTR:FINISH:=true;
if CH='C' then ALTERNOVTAP
else CHECKCROSSobstacle(GROUND)
  end;
%***********************************************\
  procedure CUMBINEFACtor;
  %***********************************************\
forward;
%***********************************************\
  procedure GROUNDFACTor(GROUND:RANGE);
  (* decides the type of action to be taken for ground GROUND *)
  %***********************************************\
  begin
case GROUND of
      1,2:REPORTWATERobstacle(GROUND);
      3:
      if ((TPTR^.VALUE1 # DIRECTION) or
     (TPTR^.VALUE2 # DIRECTION)) then
        begin
      UPKM:=2;FACTOR:=FACTOR*0.6;
        end;
      4:FACTOR:=FACTOR*0.5;
      5:
        begin
      MSG(12);
      WRITELN(TTY,'FROM LOCATION ',X2:2,Y2:2);
      WRITELN(TTY,'  TANK HAS RUN INTO ENEMY MINEFIELD!:');
      REACHEDESTination:=true;TNKDAT[IK,JK].STATUS.
      NUETRALISED:=true
        end;
      6:
        begin
      MSG(12);
      WRITE(TTY,'  FROM LOCATION ',X2:2,Y2:2);
      WRITELN(TTY,'  TANK BOGGED DOWN IN MARSHY GROUND!');
      PTR^.STATE:=false;with TNKDAT[IK,JK].STATUS do
        begin BOGGED:=true;CANMOV:=false;MOVING:=false
        end;
```

```
          REACHDESTination:=true
            end;
          T:=GROUNDFACtor
     end
       end;
     %*************************************************\
       procedure CONSTFACtor;
       %*************************************************\
       var
     I:integer;TEMP:GRIDPTR;BRIDGE:boolean;
       begin
     TEMP:=TPTR;TPTR:=TPTR^.NEXTPTR;
     BRIDGE:=false;
     if ((TEMP^.VALUE2=3)and((TEMP^.VALUE1=1) or (TEMP^.VALUE1=2)))
     then BRIDGE:=true;
     if not BRIDGE then GROUNDFACTor(TEMP^.VALUE1);
     if not FINISH then
       begin
          if TEMP^.VALUE1 < 3 then for I:=1 to 2 do TPTR:=TPTR^.NEXTPTR
          else TPTR:=TPTR^.NEXTPTR;
          GROUNDFACTor(TEMP^.VALUE2)
       end
       end;
     %*************************************************\
       procedure CHECKSIDE;
       %*************************************************\
       var
     TEMPDIR:RANGE;TEMPTR:GRIDPTR;TEMPGROUND:RANGE;I:integer;
       begin
     TEMPGROUND:=GRIDCHAR[X2,Y2].TYPOFGROUND;
     TEMPTR:=GRIDCHAR[X2,Y2].GROUNDPTR;
     TPTR:=TEMPTR;
     if TEMPGROUND # 8 then
       begin
          if TEMPGROUND # 7 then
       begin
          TEMPDIR:=(DIRECTION+2)mod(8);
          if ((TEMPTR^.VALUE1=TEMPDIR)or(TEMPTR^.VALUE2=TEMPDIR))
          then GROUNDFACTor(TEMPGROUND)
       end
          else
       begin
          TPTR:=TPTR^.NEXTPTR;
          if ((TEMPTR^.NEXTPTR^.VALUE1=TEMPDIR)or
         (TEMPTR^.NEXTPTR^.VALUE2=TEMPDIR))
          then GROUNDFACTor(TEMPTR^.VALUE1);
          TEMPGROUND:=TEMPTR^.VALUE2;
          if TEMPTR^.VALUE1 < 3 then
          for I:=1 to 2 do TPTR:=TPTR^.NEXTPTR
          else TPTR:=TPTR^.NEXTPTR;
          if ((TEMPTR^.VALUE1=TEMPDIR)or(TEMPTR^.VALUE2=TEMPDIR))
          then GROUNDFACTor(TEMPGROUND)
       end
       end
       end;
     %*************************************************\
```

```
    procedure GRADIENTfactor;
    (* calculates the gradient of move and its effect on speed *)
    \***********************************************************\
    var
 GRADIENT:integer;
    begin
 GRADIENT:=GRIDCHART[X2,Y2].ALTITUDE - GRIDCHART[X1,Y1].ALTITUDE;
 if GRADIENT > 0 then
 FACTOR:=FACTOR*(GRADIENT/50)*0.05
    end;
 \***********************************************************\
    procedure UPDATEFUEL;
    (* UPDATES THE FUEL BALANCE AFTER REDUCING CALCULATING AND THEN REDUCING IT *)
    (* updates the fuel that has been spent *)
    \***********************************************************\
    var
 BAL,FUEL:integer;
    begin
 FUEL:=TRUNC((DISTANCE*USAGE/1000)+0.5);
 BAL:=TANKDAT[TK,JK].FUELCAP - FUEL;
 if BAL <= 0 then
    begin
         MSG(12);MSG(13);
         FINISH:=true;PTR^.STATE:=false
    end
 else
    begin
         TANKDAT[K,JK].FUELCAP:=BAL;
         if FIRSTIME then
         if BAL < TANKCHAR[TYPEOFTANK].FIGURES then
    begin
         MSG(12);MSG(14);
         FIRSTIME:=false
    end
    end
    end;
    begin
 X1:=PTR^.LOC.X;Y1:=PTR^.LOC.Y;
 SPEED:=TANKCHAR[1].SPEEDROAD;
 FACTOR:=1.0;
 DISTANCE:=500;
 case DIRECTION of
 1,3,5,7:
   begin
         X2:=X1;Y2:=Y1;
         CALLNEXTGRID(X2,Y2,DIRECTION-1);
         CHECKSIDE;DISTANCE:=700;
   end;
 others:
 end;
 if not FINISH then
   begin
 X2:=X1;Y2:=Y1;
 CALLNEXTGRID(X2,Y2,DIRECTION);
 TPTR:=GRIDCHART[X2,Y2].GRIDPTR;
```

```
        GROUNDFACTOR(GRIDCHAR(X2,Y2),TYPE,GROUND);
        GRADIENTFACTOR;
          end;
        if ((not FINISH )or(CROSSED))then
          begin
        PTR^.LOC.X:=X2;PTR^.LOC.Y:=Y2;
        SPEED:=TRUNC(SPEED* FACTOR);
        TNKDAT[IK,JK].MAPCORD.X:=X2;TNKDAT[IK,JK].MAPCORD.Y:=Y2;
        CURRENTIME:=TRUNC((36*DISTANCE)/(SPEED*10));
        UPDATEFUEL;TNKDAT[IK,JK].ORIENTATION:=DIRECTION
          end
          end;
    begin
        TIMECOUNT:=0;FINISH:=false;FIRSTTIME:=true;
        CHANGEDIRECTION;
        SKIP(3);CROSSED:=false;
        IK:=(PTR^.TANKID)div(10);JK:=(PTR^.TANKID)mod(10);
        CURRENTIME:=0;
        repeat
        GETDIRECTION(PTR^.LOC.X,NEXTX,PTR^.LOC.Y,NEXTY,DIRECTION);
        COMPUTETIME;
        TIMECOUNT:=TIMECOUNT+CURRENTIME;
        if (PTR^.LOC.X=PTR^.DEST.X)and(PTR^.LOC.Y=PTR^.DEST.Y)then
          begin
        FINISH:=true;REACHEDESTination:=true;TNKDAT[IK,JK].STATUS.MOVING:=false
          end;
        if not FINISH then
        if (NEXTX=PTR^.LOC.X)and(NEXTY=PTR^.LOC.Y) then
          begin
        PTR^.CURROBJECTive:=PTR^.CURROBJECTive+1;
        CHANGEDIRECtion
          end
        until ((TIMESTEP<(TIMECOUNT+0.5*CURRENTIME))or(FINISH)or(REACHEDESTination));
    end;
        %*****************************************************************************\
    procedure MOVETANKS;
    (*moves the tanks from a location to another when command has been given *)
    %*****************************************************************************\
    var
        TEMP,PTR:MPTR;
    begin
        PTR:=MOVTABROOT^.LINK;
        while PTR#nil do
          begin
        REACHEDESTination:=false;
        if PTR^.STATE then PROCESSMOVEtank(PTR);
        TEMP:=PTR;
        PTR:=PTR^.LINK;
        if REACHEDESTination then DELETEMOVEtab(TEMP)
          end
    end;
        %*****************************************************************************\
    procedure FIRE;
    (*FIRES the shell that has been commanded for the tank to fire *)
    %*****************************************************************************\
    var
```

```
        J,PCHS,DISTANCE,ND,PENPOTST,HITX,HITY:integer;
        PROB,ASINE,ACOS:real;
        LOCHIT:boolean;
        %***************************************************************\
   procedure CHECKHIT;
   (* CHECKS FOR THE EFFECT OF THE FIRING OF THE SHELL AND TAKES ACTION ACCORDINGLY *)
   %*****************************************************************\
          var
        II,JJ:integer;
        %***************************************************************\
          procedure ANALYISEHIT(I,J,HITYPE,AMMTYPE:integer);
          (* analyises the hit in terms of its angle of hit,distances and where it hit*)
          %**************************************************************\
          var
        DISTYPE,ANGLETYPE,ANGLE:integer;
          function PENETRATED:boolean;
          var
               TYP,GUNSHOT,ARMR1,ARMR2,ARMR3:integer;
          begin
               ARMR1:=TANKCHAR[TNKDAT[I,J].TYPNO].ARMRFRONT;
               ARMR2:=TANKCHAR[TNKDAT[I,J].TYPNO].ARMRSIDE;
               ARMR3:=TANKCHAR[TNKDAT[I,J].TYPNO].ARMRREAR;
               TYP:=TNKDAT[I,J].TYPNO;
               PENETRATED:=false;
               if AMMTYPE=3 then
          begin
               case DISTYPE of
               0:GUNSHOT:=GUNCHAR[TYP].AR500;
               1:GUNSHOT:=GUNCHAR[TYP].AR1000;
               2:GUNSHOT:=GUNCHAR[TYP].AR2000;
               others:
               end;
               case ANGLE of
               1:
               if GUNSHOT> ARMR1 then PENETRATED:=true;
               2:
               if GUNSHOT>ARMR2 then PENETRATED:=true;
               3:
               if GUNSHOT>ARMR3 then PENETRATED:=true
               end;
          end;
          end;
        %****************************************************************\
          procedure        REPORTFIRE(I1,J1,AMN:integer);
          (*reports coming under fire giving the type of fire that is there *)
          %*************************************************************\
          var
               TYP:packed array[1..5]of char;
          begin
               if FIREREPORT[I1,J1] = true then
          begin
               case AMN of
               1:TYP:='SMOKE';
               2:TYP:='HE   ';
               3:TYP:='AP   ';
               end;
```

```
            SKIP(2);FIREREPORT[I,J]:=true;
            WRITELN(TTY,'TANK',I:1,J:1,'REPORTS [illegible] FROM',TYP,'FIRE [illegible] LOC'
              ,FR^.
              SOURCELOC.X:2,
              FR^.SOURCELOC.Y:2);
     end;
     end;
%***********************************************************************\
     procedure INITREPORTfire;
     (* initialises the report for tanks so that in the same burst
       the fire is not reported by the same tank more than once *)
     %***********************************************************************\
     var
            I,J:integer;
     begin
            for I:=1 to 5 do
            for J:=1 to 3 do
              begin   FIREREPORT[I,J]:=false;
            HIT[I,J]:=false;
              end;
            LOCHIT:=false;
     end;
%***********************************************************************\
     procedure REPORTHIT(I,J:integer);
     (* in case the tank target has been hit this reported in this procedure *)
     %***********************************************************************\
     begin
            if not HIT[I,J] then
     begin
            TNKDAT[I,J].STATUS.NEUTRALISED:=true;
            WRITELN(TTY,'TANK ',FR^.SOURCETK[1]:1,FR^.SOURCETK[2]:1,
            ' REPORTS NEUTRALISING ENEMY TK=> LOC ',HITX:2,HITY:2);
            BREAK;HIT[I,J]:=true;
     end;
     end;
     begin
 if ((I#IK)and(J#JK)) then
     begin
            REPORTFIRE(I,J,AMMTYPE);
            if ((AMMTYPE#1)or(HITTYPE#1)) then
     begin
            if DISTANCE < 1000 then DISTYPE:=0
            else
            if DISTANCE <2000 then DISTYPE:=1
            else
            if DISTANCE <4000 then DISTYPE:=2
            else DISTYPE:=3;
            ANGLE:=ABS(TNKDAT[IK,JK].ORIENTATION-TNKDAT[IK,JK].ORIENTATION);
            if
            ((ANGLE < 45)or(ANGLE >315)) then ANGLETYPE:=0
            else
            if ((ANGLE>135)and(ANGLE<225)) then ANGLETYPE:=1
            else ANGLETYPE:=2;
            if PENETRATED then
            REPORTHIT(I,J)
            else
```

```
22970                   begin
22980               case OBJTYPE of
22990               0:
23000                   if ANGLETYPE=2 then
23010                 begin
23020                   case ANGLETYPE of
23030                   1:with PKDAT[I,JJ].STATUS do
23040                     begin
23050                       CANSHOOT:=false;
23060                       SHOCKED:=true;
23070                     end;
23080                   2:PKDAT[I,JJ].STATUS.CANMOV:=
23090                   false;
23100                   3:REPORTHIT(I,J)
23110                   end
23120                 end
23130                   else PKDAT[I,JJ].STATUS.SHOCKED:=true;
23140               1:
23150                   if ANGLETYPE=2 then
23160                 begin
23170                   case ANGLETYPE of
23180                   1:with PKDAT[I,JJ].STATUS do
23190                     begin
23200                       CANSHOOT:=false;
23210                       SHOCKED:=true;
23220                     end;
23230                   2:PKDAT[I,JJ].STATUS.CANMOV:=
23240                   false;
23250                   3:REPORTHIT(I,J)
23260                   end
23270                 end
23280                   else
23290                   if ANGLETYPE=2 then PKDAT[I,JJ].STATUS.
23300                   CANMOV:=false
23310                   else
23320                 begin
23330                   PKDAT[I,JJ].STATUS.SHOCKED:=true;
23340                   PKDAT[I,JJ].STATUS.CANSHOOT:=false
23350                 end;
23360               2:
23370                   if ANGLETYPE=3 then PKDAT[I,JJ].STATUS.
23380                   CANMOV:=false
23390                   else
23400                   PKDAT[I,JJ].STATUS.SHOCKED:=true;
23410               others:
23420               end;
23430                 end;
23440           end;
23450           end;
23460           end;
23470           begin
23480       for II:=1 to 5 do
23490       for JJ:=1 to 3 do
23500         begin
23510             if ((PKDAT[II,JJ].MAPCORD.X=HITX)and
23520             (PKDAT[II,JJ].MAPCORD.Y= HITY)) then
```

```
                ANALYISEHIT(II,JJ,0,FR^.TYPOFAM)
                else
                if ((ABS(TSKDAT[II,JJ].ARCORD.X-HITX)=1)and
                (ABS(TSKDAT[II,JJ].ARCORD.Y-HITY)= 1)) then
                ANALYISEHIT(II,JJ,1,FR^.TYPOFAM);
            end;
        if FR^.DESTK[1]=0 then
        if ((HITX=FR^.DESTLOC.X)and(HITY=FR^.DESTLOC.Y)and(not LOCHIT)) then
            begin
        WRITELN(TTY,'TASK',FR^.SOURCETK[1]:1,FR^.SOURCETK[2]:1,
        'REPORTS GIVING EFFECTIVE FIRE ON ',HITX:4,HITY:2);
        LOCHIT:=true;
            end;
            end;
        %*****************************************************************************\
procedure INITREPORTfire;
(* initialises the report fire to false *)
%***************************************************************************\
            var
        I,J:integer;
            begin
        for I:=1 to 5 do
        for J:=1 to 3 do FIREREPORT[I,J]:=false
            end;
        %*****************************************************************************\
procedure CALCULATERounds(FTEMP:FPTR;var ROUNDS:integer);
(* calculates the rounds that are to be fired in that timestep *)
%***************************************************************************\
            begin
        if FTEMP^.TIME < MINUTESTEP then
            begin
        ROUNDS:=(MINUTESTEP-FTEMP^.TIME)*FTEMP^.RATE;
        FTEMP^.TIME:=0;
        if ROUNDS > FTEMP^.TOTNUM then ROUNDS:=FTEMP^.TOTNUM
            end
        else
            begin
        FTEMP^.TIME:=FTEMP^.TIME - MINUTESTEP;
        ROUNDS:=0
            end
            end;
begin
        FR:=FROOT;REWRITE(FIREOUT);
        while FR^.LLINK<>nil do
            begin
        FR:=FR^.LLINK;CALCULATERounds(FR,NO);INITREPORTfire;LOCHIT:=false;
        if NO >0 then
            begin
        PLUS:=1;TSKDAT[FR^.SOURCETK[1],FR^.SOURCETK[2]].STATUS.FIRING:=true;
        CALCULATEDist(FR^.SOURCELOC.X,FR^.SOURCELOC.Y,FR^.DESTLOC.X,FR^.DESTLOC.
                Y,DISTANCE);
        ASINE:=(FR^.DESTLOC.Y - FR^.SOURCELOC.Y)/DISTANCE;
        ACOS:=(FR^.DESTLOC.X - FR^.SOURCELOC.X)/DISTANCE;
        for J:=1 to NO do
            begin
                if PLUS=1 then PROB:=2- FR^.HITPROB
```

```
                else PROB:=FR^.HITPROB;
                TEMPDIST:=TRUNC(PROB*DISTANCE+0.5);
                HITX:=FR^.SOURCEGOC.X+TRUNC(TEMPDIST*ACOS+0.5);
                HITY:=FR^.SOURCEGOC.Y+TRUNC(TEMPDIST*ASINE+0.5);
                WRITELN(FIRED?,HITX,HITY);
                PLUS:=PLUS * (-1);
                FR^.HITPROB:=SQRT(FR^.HITPROB);
                CHECKHIT;
             end;
          FR^.TOTNUM:=FR^.TOTNUM-10;
          case FR^.TYPOFAM of
                1:TNKDAT[FR^.SOURCETK[1],FR^.SOURCETK[2]].AMMCAP.TYP1:=TNKDAT[FR^.
                SOURCETK[1],FR^.
                SOURCETK[2]].AMMCAP.TYP1-10;
                2:TNKDAT[FR^.SOURCETK[1],FR^.SOURCETK[2]].AMMCAP.TYP2:=TNKDAT[FR^.
                SOURCETK[1],FR^.
                SOURCETK[2]].AMMCAP.TYP2-10;
                3:TNKDAT[FR^.SOURCETK[1],FR^.SOURCETK[2]].AMMCAP.TYP3:=TNKDAT[FR^.
                SOURCETK[1],FR^.
                SOURCETK[2]].AMMCAP.TYP3-10
          end;
          if FR^.TOTNUM=0 then
             begin
                FQ:=FR;
                TNKDAT[FR^.SOURCETK[1],FR^.SOURCETK[2]].STATUS.FIRING:=false;
                FR:=FR^.JLINK;
                DELETEFTAB(FQ);
             end;
             end;
             end;
       end;
          %****************************************************************************\
       procedure TRANSFER(TEMP:LETTERWORD);
       (* transfers the words that are to be displayed on the graphics display in char *)
       %****************************************************************************\
       var
          I:integer;
       begin
          for I:=1 to 12 do
          NAME[I]:=TEMP[I];
          NAME[13]:='*';
          NAME[14]:='.';
       end;
          %****************************************************************************\
       procedure DRAWLANDMARK(LP:LMPTR);
       (* draws the landmarks that have been defined by the user on the graphics display *)
       %****************************************************************************\
       var
          X,Y:integer;
       begin
          if LP^.LLINK#nil then DRAWLANDMARK(LP^.LLINK);
          if LP#LMROOT then
             begin
          CONVERT(LP^.GR,X,Y);TRANSFER(LP^.NAME);
          LINE(REDUCE(X),REDUCE(Y),0);
          STRING(NAME);
```

```
        end;
      if UP^.RLINK#nil then DRAWLANDMARK(UP^.RLINK);
   end;
      %**************************************************************************\
   procedure DRAWTANK;
   (* draws the tanks at the location they are at that time, giving the tank nos *)
   %**************************************************************************\
   var
      I,J:integer;
   begin
      TNAME[3]:='*';
      TNAME[4]:='.';
      for J:=1 to 5 do
        begin
      TNAME[1]:=CHR(ORD('0')+J);
      for I:=1 to 3 do
        begin
      TNAME[2]:=CHR(ORD('0')+I)
      ;LINE(REDUCE(TNKDAT[I,J].HARDORD.X),REDUCE(TNKDAT[I,J].HARDORD.Y),0);
      STRING(TNAME);
        end;
        end;
   end;
      %**************************************************************************\
   procedure DRAWFIREPT;
   (* draws the location of hit of the shells *)
   %**************************************************************************\
   var
      X,Y:integer;
   begin while EOF(FIREPT) do
        begin
      READLN(FIREPT,X,Y);
      LINE(REDUCE(X),REDUCE(Y),0);
      MARKER(8);
        end
   end;
      %**************************************************************************\
   procedure DRAWMAP;
   (* draws the map on the graphic display *)
   %**************************************************************************\
   var
      XX,YY:real;
      W,SP,D,H:integer;X,Y,X1,Y1,MARK:integer;
      XINCR,YINCR:real;
      LASTX,LASTY:real;FIRST:boolean;
   procedure READINPUT;(*READS DIRECTLY INPUT UPTO 12 CHARACTERS*)
   (* reads directly 12 char from file MAPDAT and stores them in WORD.
    used to interactive communication *)
   %**************************************************************************\
        var
      INCH:char;I:integer;
        begin
      for I:=1 to 12 do WORD[I]:=' ';
      INCH:=' ';I:=1;
      while INCH=' ' do READ(MAPDAT,INCH);
      while ((not EOLN(MAPDAT))and(I<13)and(INCH#',')) do
```

```
25210           begin
25220         MAP0[I]:=INCH;I:=I+1;READ(MAPDAT,INCH);
25230           end;
25240         if I>12 then MSG(5)
25250         else
25260         if EOLN(MAPDAT) then MAP0[I]:=INCH;
25270         while not EOLN(MAPDAT) do READ(MAPDAT,INCH);
25280           end;
25290         %*****************************************************************\
25300       procedure DRAW;
25310       (* draws the details of the map as defined earlier *)
25320       %*****************************************************************\
25330           var
25340         LASTX,LASTY:integer;
25350         %*************************************************************\
25360           procedure FILLBETWEEN(X,LASTX,Y,LASTY:integer);
25370           (* used for get inbetween points for the display purpose *)
25380           %*********************************************************\
25390           var
25400         PX,PY:real;TEMP:integer;
25410           begin
25420         TEMP:=LASTX-X;PX:=X;PY:=Y;
25430         if TEMP=1 then PX:=X+0.5
25440         else
25450         if TEMP=(-1) then PX:=LASTX+0.5;
25460         TEMP:=LASTY-Y;
25470         if TEMP=1 then PY:=Y+0.5
25480         else
25490         if TEMP=(-1) then PY:=LASTY+0.5;
25500         if ((X<>LASTX)or(Y<>LASTY)) then
25510           begin
25520             LINE(PX/40,2*Y/40,0);MARKER(MARK);
25530           end;
25540           end;
25550           begin
25560         loop
25570         READINPUT;
25580         NUMBER;
25590         exit if not ISNUMBER;
25600         CONVERT(INTVAL,X1,Y1);
25610         LASTX:=X;LASTY:=Y;
25620         repeat
25630               READLN(MAP,X,Y);
25640               FILLBETWEEN(X,LASTX,Y,LASTY);
25650               LINE(REDUCE(X),REDUCE(Y),0);
25660               MARKER(MARK);
25670               LASTX:=X;LASTY:=Y;
25680         until((X=X1)and(Y=Y1));
25690         end
25700           end;
25710         %*****************************************************************\
25720       procedure READCHAR;(*READS A CHARACTER FROM THE FILE MAPDAT*)
25730       %*****************************************************************\
25740           var
25750         ALP:char;
25760           begin
```

```
25770             repeat READ(MAPDAT,CH)
25780             until CH#' ';
25790             while not (EOLN(MAPDAT)or(ALPH='.')) do READ(MAPDAT,CH);
25800               end;
25810         begin
25820             RESET(MAPDAT);RESET(MAP);
25830             WRITELN(TTY,'GIVE POINT FOR START MAP->');BREAK;READLN(TTY);
25840             RESET(FIRED);CLRDEV(1);
25850             LINE(0.0,0.0,0);
25860             LINE(1.0,0.0);
25870             LINE(1.0,1.0);
25880             LINE(0.0,1.0);
25890             LINE(0.0,0.0);
25900             YY:=0.0;
25910             CSIZEH(120);CSIZEV(60);
25920             for I:=1 to 7 do
25930               begin
25940             XX:=0.0;
25950             YY:=YY+0.125;
25960             for J:=1 to 7 do
25970               begin
25980             XX:=XX+0.125;
25990             LINE(XX,YY,0);
26000             MARKER(3);
26010               end;
26020               end;
26030             for I:=1 to 7 do
26040               begin
26050             loop
26060             READCHAR;
26070             exit if CH#'Y';
26080             if I<3 then
26090               begin
26100                 READLN(MAPDAT,N);READLN(MAPDAT,D);
26110               end
26120             else
26130             if I=7 then READLN(MAPDAT,N);
26140             READLN(MAPDAT,SP);
26150             CONVERT(SP,X,Y);
26160             LINE(REDUCE(X),REDUCE(Y),0);
26170             FIRST:=true;
26180             if ((I=1)or(I=3)) then
26190               begin
26200                   loop
26210                   READINPUT;NUMBER;
26220                   exit if not ISNUMBER;CONVERT(INPVAR,X1,Y1);
26230                   LINE(REDUCE(X1),REDUCE(Y1),1);XINCR:=0.0;YINCR:=0.0;
26240                   if I=1 then
26250                     begin
26260                   if X=X1 then XINCR:=-0.005
26270                   else YINCR:=0.005;
26280                   if FIRST then
26290                     begin FIRST:=false;LASTX:=REDUCE(X)+XINCR;LASTY:=
26300                   REDUCE(Y)+YINCR;
26310                     end;
26320                   LINE(LASTX,LASTY,0);
```

```
                LASTX:=REDUCE(X1)+XINCR;LASTY:=REDUCE(Y1)+YINCR;
                LINE(LASTX,LASTY,1);
                LINE(REDUCE(X1),REDUCE(Y1),0);
                  end;
                X:=X1;Y:=Y1;
                end;
            end
        else
            begin
                case I of
                1,3:WRITELN(TTY,'ERROR');
                2:MARK:=3;
                4:MARK:=1;
                5:MARK:=4;
                6:MARK:=5;
                others:MARK:=10
                end;
                MARKER(MARK);
                DRAW;
            end;
        end
          end;
        DRAWLANDMARK(LMRDOF);DRAWTANK;
        DRAWFIREPT;
        WRITELN(TTY,'GIVE <RETURN> TO CLEAR & CONTINUE');READLN(TTY);
        CLRDEV(1);
end;
        %***************************************************************************\
procedure COMNDPROS;(*PROCESSES THE TYPE OF COMMAND*)
%***************************************************************************\
begin READWORD;
    if WORD='REPORT         ' then REPORTPROC
    else
    if WORD='MOVE           ' then MOVEPROCEDURE
    else
    if ((WORD='FIRE         ')or(WORD='SHOOT          '))then FIREPROCEDURE
    else
        begin
    WRITELN(TTY,'YOUR STATEMENT':20,WORD:15,'NOT RECOGNISED PLEASE RETYPE':30);
    BREAK
        end
end;
        %***************************************************************************\
procedure TANKNAME;(*READS THE NAME OF TANK ADRESSED*)
(* the no of the tank is stored in IK=tankno,JK=troopno.*)
(* it then checks if the tank is nuetralised .*)
(*if it is NEUTRALISED  then IT  is not allowed to be accesed *)
%***************************************************************************\
begin
    READWORD;
    NUMBER;
    TANKNUMBER:=INTVAL;
    IK:=NUMWORD[1];JK:=NUMWORD[2];
    if ((IK<6)and(IK>0)and(JK>0)and(JK<4)) then
        begin
    if ((not TNKDAT[IK,JK].STATUS.NUETRALISED)and(not TNKDAT[IK,JK].STATUS.SHOCKED))
```

```
26890          then COMMDERUS;
26900            end
26910          else WRITELN(TTY,'TANK NO ',IK:1,JK:1,' WRONG GIVE AGAIN');
26920      end;
26930        {*****************************************************************\
26940      procedure CHECK;
26950      (* analyises the tyoe of CHECK required and takes actio accordingly *)
26960      {*****************************************************************\
26970      begin WRITELN(TTY,'THE FOLLOWING CHECKS ARE POSSIBLE ');
26980          WRITELN(TTY,'  ':15,'LISTING LANDMARKS TYPE "L"');
26990          WRITELN(TTY,'  ':15,'LISTING CURRENTLY MOVING TANKS-"T"');
27000          WRITELN(TTY,'  ':15,'DRAWINGG THE MAP "M"');
27010          WRITELN(TTY,'  ':15,'SETTING TO DRAW THE MAP-"S"');
27020          WRITELN(TTY,'  ':15,'DELETING THE SET MAP DRAWING-"D"');
27030          WRITELN(TTY,'  ':15,'RETURN-"R"');WRITE(TTY,'CHECK REQUIRED FOR:');
27040          BREAK;READCH;
27050          case CH of
27060          'L':PRILMTAB(LMROOT);'T':PRINTMOVTAb;
27070          'M':DRAWMAP;'S':MAPREQUIREd:=true;'D':MAPREQUIREd:=false;
27080          others:
27090            begin WRITELN(TTY,'YOUR ',CH,' NOT RECOGNISED PLEASE RETYPE');
27100              CHECK
27110            end
27120          end;
27130      end;
27140      begin
27150          INITIALISE;
27160          INITGRID;
27170          INITTANK;
27180          FILLTOPOGRaphy;
27190          ENTERLM;
27200          READTIMEStep;
27210          loop
27220          SKIP(1);
27230          if not FREEZE then
27240            begin  MOVETANKS;REMOVESHOCk;FIRE;
27250            end;
27260          if MAPREQUIREd then DRAWMAP;
27270          WRITE(TTY,'YOUR COMMAND:');BREAK;
27280          UPDATETIME;
27290          READLINE;READWORD;TESTNUMBER:=false;
27300          exit if WORD='END       ';
27310          if WORD='TK          ' then
27320          TANKNAME
27330          else
27340          if (WORD='HELPCOMND   ')or (WORD='HELP        ') then HELPCOMND
27350          else
27360          if WORD='LANDMARK    ' then ENTERLM
27370          else
27380          if WORD='CHECK       ' then CHECK
27390          else
27400          if WORD='FREEZE      ' then FREEZE:=true
27410          else
27420          if ((WORD='CONTINUE    ')or(WORD='CONT        '))then FREEZE:=
27430          false
27440          else ERROR(1)
```